ANL-ART-131

Fast Reactor Methods Development Plan

Nuclear Science and Engineering Division

DOCUMENT AVAILABILITY

Online Access: U.S. Department of Energy (DOE) reports produced after 1991 and a growing number of pre-1991 documents are available free via DOE's SciTech Connect (http://www.osti.gov/scitech/)

Reports not in digital format may be purchased by the public from the National Technical Information Service (NTIS):
U.S. Department of Commerce
National Technical Information Service
5301 Shawnee Rd
Alexandria, VA 22312
www.ntis.gov
Phone: (800) 553-NTIS (6847) or (703) 605-6000
Fax: (703) 605-6900
Email: orders@ntis.gov

Reports not in digital format are available to DOE and DOE contractors from the Office of Scientific and Technical Information (OSTI):
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831-0062
www.osti.gov
Phone: (865) 576-8401
Fax: (865) 576-5728
Email: reports@osti.gov

ANL-ART-131

Fast Reactor Methods Development Plan

prepared by
T. H. Fanning, M. A. Smith, A. J. Brunett

Nuclear Science and Engineering Division
Argonne National Laboratory

June 30, 2018

EXECUTIVE SUMMARY

Fast reactor design and safety analysis methods (computer codes) have been under development for approximately sixty years. During the first forty years, code development was driven by the immediate needs of operational facilities such as Experimental Breeder Reactors I and II and the Fast Flux Test Facility. Development was further driven by programmatic needs to support advanced reactor programs such as the Clinch River Breeder Reactor Project and the Advanced Liquid Metal Reactor Program. By the early 1990s, design and safety analysis codes had been well established to addressed the unique characteristics of fast reactors.

During the subsequent twenty years, however, development was nearly stagnant as a result of U.S. Government policy changes that abruptly canceled nearly all of the fast reactor research in the United States. As a result, the state of fast reactor design and safety analysis methods has deteriorated.

> *GOAL:*
>
> *To ensure the existing fast reactor design and safety analysis codes are available to adequately support a fast reactor authorization basis or license application.*

This report defines a Fast Reactor Methods Development Plan that describes the scope of maintenance and development activities that are required to ensure the existing fast reactor design and safety analysis codes are available to adequately support a fast reactor authorization basis or license application. Of highest priority is the need to establish a software development team that will implement and execute quality software engineering practices to preserve the tremendous amount of knowledge and experience gained during the preceding six decades.

Design and safety analysis codes for fast reactors provide unique capabilities not available in codes designed for light-water reactors. By 1980, the number of codes available for fast reactor design and safety analysis was around 130 *in the U.S. alone*. By 2010, the number had dwindled to approximately 60 *worldwide*, with less than eighteen potentially available in the U.S. Of these, seven are identified in this report as being critical for design and safety analysis:

- MC^2 implements a detailed treatment of high-energy resonances in both the resolved and unresolved regions of nuclear data, which is essential for fast spectrum systems.
- DIF3D/VARIANT provides high-order transport solvers to account for the long mean-free path and high leakage effects in fast spectrum systems and to correctly determine flux and power distributions, control-rod worth, and shutdown margins.
- REBUS analyzes discrete and equilibrium fuel cycle scenarios to assess requirements for heavy metal loading, enrichment, depletion, fuel shuffling, mass flows, fluence limits, and other conditions for breeder, burner, or break-even core configurations.
- GAMSOR evaluates important gamma transport effects from fueled to unfueled core regions.
- SUPERENERGY performs optimization of flow orifice design strategies for ducted fuel assemblies at all stages of the fuel cycle to account for thermal margins and uncertainties.
- PERSENT implements perturbation and sensitivity analysis capabilities to determine fast spectrum kinetic parameters and reactivity feedback effects including the impacts of geometric deformation, material expansion and relocation, and the Doppler broadening of high-energy resonances.
- SAS simulates anticipated operational occurrences, design-basis accidents, beyond design-basis events, and design extension conditions — accounting for inherent fast spectrum

feedback effects and passive safety features — to assess margins for structural thermal limits, metallic fuel failure, sodium boiling, and fission product release.

The expertise represented by these codes was gained through real-time collaboration between code developers and facility analysts (users) over an extended period of time. During that time, verification and validation were driven by user need and were an integral part of the development process.

This report does *not* define a verification and validation plan. From the perspective of the developer, verification ensures that the software conforms to the defined analytical requirements and satisfies its intended use. From the perspective of the *user*, verification and validation relates to the *application* of a given computer code to a particular *design*. The computer codes documented in herein are intended to be generally applicable to a wide range of fast spectrum systems, but it is not practical for the developers to anticipate all conceivable design configurations. End users are expected to define a verification and validation plan according to their requirements. Should deployment of a sodium fast reactor become a priority, programmatic needs would again drive the integration of software verification and model validation.

Based on the broad, high-level goal stated above, specific goals are defined and include the following:

- Implement Software Quality Assurance Program
- Document Code Theory, Implementation, and Use
- Establish Software Configuration Management
- Ensure Compliance with Modern Code Standards
- Maintain Cross-Platform Compatibility
- Identify and Resolve Critical Modeling Gaps
- Provide a User Support Environment

For each code, a detailed assessment against each of these goals is documented. From these assessments, critical gaps are identified and documented. The most critical gap overall is the lack of a software development team to support code maintenance under a software quality assurance program. If a team and program were in place, technical gaps could be prioritized and closed. At the same time, a user support environment should be established to include formal issue reporting and response mechanisms and training opportunities to help preserve the analysis knowledge base.

RECOMMENDATIONS:

- *Establish a Development Team*
- *Implement Software Quality Assurance*
- *Create a User Support Environment*
- *Close Critical Modeling Gaps:*
  - *Implement missing transient phenomena for metal fuel in SAS*
  - *Eliminate outdated memory management from DIF3D-based codes*
  - *Improve compatibility and documentation for REBUS*

These recommendations only address the most critical gaps. However, implementing these recommendations will ensure the existing fast reactor design and safety analysis codes remain viable. Other important gaps exist that should be evaluated and prioritized based on future programmatic needs.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 Introduction

Development and deployment of *advanced fast reactors*, whether authorized by the U.S. Department of Energy or licensed by the U.S. Nuclear Regulatory Commission, requires the application of various analysis methods (computer codes) to demonstrate that design-basis performance objectives are met and that system responses will satisfy regulatory requirements for all anticipated operational occurrences, design-basis events, and design extension conditions. This report provides an assessment and identifies critical gaps for selected fast reactor analysis codes that are an essential part of the design and safety analysis workflow. Critical gaps are categorized according to a number of high-level goals. The results of the assessment are then used to define priorities for improving each of the analysis codes according to these goals.

Prior work that assessed safety and licensing barriers has been leveraged for this report. Sandia National Laboratories (SNL) led the development of a broad "Sodium Fast Reactor Safety and Licensing Research Plan" (SLRP) [1] [2] that identified potential research priorities for the U.S. DOE based on existing gaps in the safety case that would be used to satisfy the license application for a sodium-cooled fast reactor (SFR). Of the five topical areas addressed in the study, the highest priority that required near-term resolution was the suitability of existing codes and models to support a license application. A subsequent gap analysis by Argonne National Laboratory reiterated the findings of the SNL study, and indicated that resolution of this item had high regulatory significance. [3]

> *At the highest level, the panel judged that current US code capabilities are adequate for licensing given reasonable margins, but expressed concern that US code development activities had stagnated and that the experienced user-base and the experimental validation base was decaying away quickly.*
>
> *SAND2011-4145, June, 2011*

As a result of the recommendations of these reports, selected tasks were initiated under the Regulatory Technology Development Plan (RTDP) developed by the Idaho National Laboratory (INL). [4] One of the tasks was to establish software quality assurance (SQA) practices for SFR safety analysis codes that would be used in an authorization or licensing framework. [5] [6] [7] [8] This new task was limited in scope and duration. The limited scope focused on computer codes that were in active use for SFR design and safety analysis and whose maintenance was the responsibility of DOE. To fit within the duration constraints, SQA practices were implemented only for a single analysis code, SAS4A/SASSYS-1. Furthermore, the task did not address potential gaps in code capabilities or phenomenological models.

This work summarizes the evaluations made in previous work, defines goals that need to be achieved to ensure stable methods development activities, and also identifies the most important maintenance and modeling gaps that need to be addressed to close the identified gaps. To be clear, these modeling gaps do not prohibit the development and deployment of new advanced fast reactors, but they may impose limitations on the scope of the analyses that can be done. In some cases, these limitations are resolved through simple approximations with bounding assumptions and larger uncertainties. In other cases, modeling updates that remove the limitation are available from other programs, but resources are not available to integrate the updates into the current workflow.

Finally, this work recommends that DOE establish development teams funded in a manner to ensure programmatic knowledge preservation and continuity of maintenance and development activities. The most critical modeling gaps are also identified, and these gaps should be the first ones addressed by the developers.

2 Background

In the late 1960s, the then U.S. Atomic Energy Commission gave development of a sodium fast reactor a high priority, and the development of the Fast Flux Test Facility (FFTF) became a cornerstone of that program. To provide adequate support for the FFTF and for the expected SFRs to follow, a major base technology program was established which provided a continuous stream of experimental information and design correlations. This experimental data would either confirm design choices or prove the need for design modifications. At the time, the "tremendous amount of data and experience pertaining to thermal design" of LMRs was recognized as providing the technical foundation for the future commercial development of LMRs. [9]

Along with the generation of experimental data came the development of safety analysis methods that used that data in correlations for mechanistic, probabilistic, or phenomenological models. These models were developed for a variety of needs ranging from individual components, such as heat exchangers, pumps, or containment barriers, to whole core or even whole-plant dynamics. A major portion of the overall technical effort since that time has been allocated to safety considerations.

In 1977, the U.S. DOE compiled a "compendium" of computer codes that were available for the safety analysis of fast reactors. [10] The report summarized the capabilities and availability of approximately 130 computer codes that were maintained *within the U.S.* by national laboratories, industry, and universities. In the development of the Safety Licensing Research Plan in 2011, experts identified approximately 60 computer codes that may be available *worldwide*. [2] Of these, roughly 18 remain within the U.S. and some of those may no longer be viable codes.

The Safety Licensing Research Plan had very broad objectives beyond evaluating the suitability of existing codes and methods to support a license application. Five topic areas were examined for safety-related gaps:

- Accident Sequences and Initiators
- Sodium Technology
- Fuels and Materials
- Source Term Characterization
- Codes and Methods

Across all five topic areas, cross-cutting gaps related to knowledge preservation were the most significant. Of the ten specific gaps identified under the Codes and Methods topic, six were ranked with high importance and a low state of knowledge. These gaps are summarized in Table 1. Two of the gaps listed, CM03 and CM10, have strong ties to the Fuels and Materials topic, while two more, CM07 and CM09, have strong ties to the Source Term Characterization topic. The remaining two, CM06 and CM08, are directly related to transient safety analysis.

In response to the findings of the Safety Licensing Research Plan, DOE funded a number of activities to begin addressing some of these gaps. Progress made in each of areas is also summarized in Table 1.

Table 1: Codes and Methods Gaps Ranked with High Importance and Low State of Knowledge. [2]

ID	*Gap Area*	*Summary of Progress*
CM03	LIFE-METAL/LIFE-4 Update	Most activities are related to application of LIFE-Metal for reactor design studies and support of licensing efforts for foreign organizations. Additional calibration and validation is needed for advanced fuels. Sensitivity screening studies have been completed for LIFE-4 to identify improvements needed to current models. [11]
CM06	Predictive Fuel Pin Failure Models	Limited funding from DOE. Most developments in this area are sponsored by the Korea Atomic Energy Research Institute (KAERI) as part of the Prototype GenIV Sodium Fast Reactor (PGSFR) development program (see Section 5.7.6). Some additional funding is provided by domestic users.
CM07	Source Term Models	Under DOE's Regulatory Technology Development Plan (RTDP), DOE funded a sequence of activities to identify relevant source-term phenomena for pool-type SFRs, to estimate release fractions from metallic fuel, and to demonstrate the feasibility of performing source-term calculations with available codes. [12] [13] [14]
CM08	SAS4A/SASSYS-1 Code Modernization	Under DOE's office of Advanced Reactor Technology (ART), code modernization activities are making significant improvements to the code and memory architecture. [15] [16] [17] [18] [19] [20] Domestic industry provides support for code improvements to support their modeling needs, and those improvements are shared with all users. Under the RTDP, developers established a formal software quality assurance plan. [7] [8]
CM09	MELCOR/CONTAIN-LMR Update	Under ART, sodium-specific models from CONTAIN-LMR have been incorporated into the MELCOR code. [21] [22] [23] [24]
CM10	Fuel Performance Documentation and Training	Under the RTDP, significant efforts have been made to establish qualification of historic metal fuel performance data and models.

3 Current Scope

3.1 Basis

The Safety Licensing Research Plan [2] provides a strong foundation for identifying issues related to fast reactor methods. For the purposes of the current work, however, there are two limitations. First, the Safety Licensing Research Plan addressed a very broad range of topics beyond methods development. As a result, the identified gaps are necessarily high-level perspectives that do not directly address methods development needs. Second, the Codes and Methods expert group assessed gaps with respect to events and accident scenarios rather than code status. This is, if a code existed to cover a particular phenomenon, then the "maturity level" (including software quality engineering, SQE) was considered to be high regardless of the status of code maintenance. The exception to this is with respect to the SAS4A/SASSYS-1 safety analysis code, for which the panel noted "that work was needed to support modernization of the code architecture, establish a more vigorous code verification and QA plan for code maintenance, configuration management/control, and testing of software through improved SQE practices."

Although the panel correctly identified the needs for code modernization and improved software quality assurance (SQA) practices for SAS4A/SASSYS-1, the same gaps exist for *all* fast reactor analysis codes. Many of the heavily relied-upon codes for design and safety analysis of fast reactors were under rigorous development and configuration control up until 1994, when U.S. Government policy changes abruptly canceled nearly all of the fast reactor research in the United States. In the subsequent two decades, almost no resources were dedicated to preserving the design and safety analysis capabilities that had been established during the preceding four decades.

3.2 Key Functional Areas

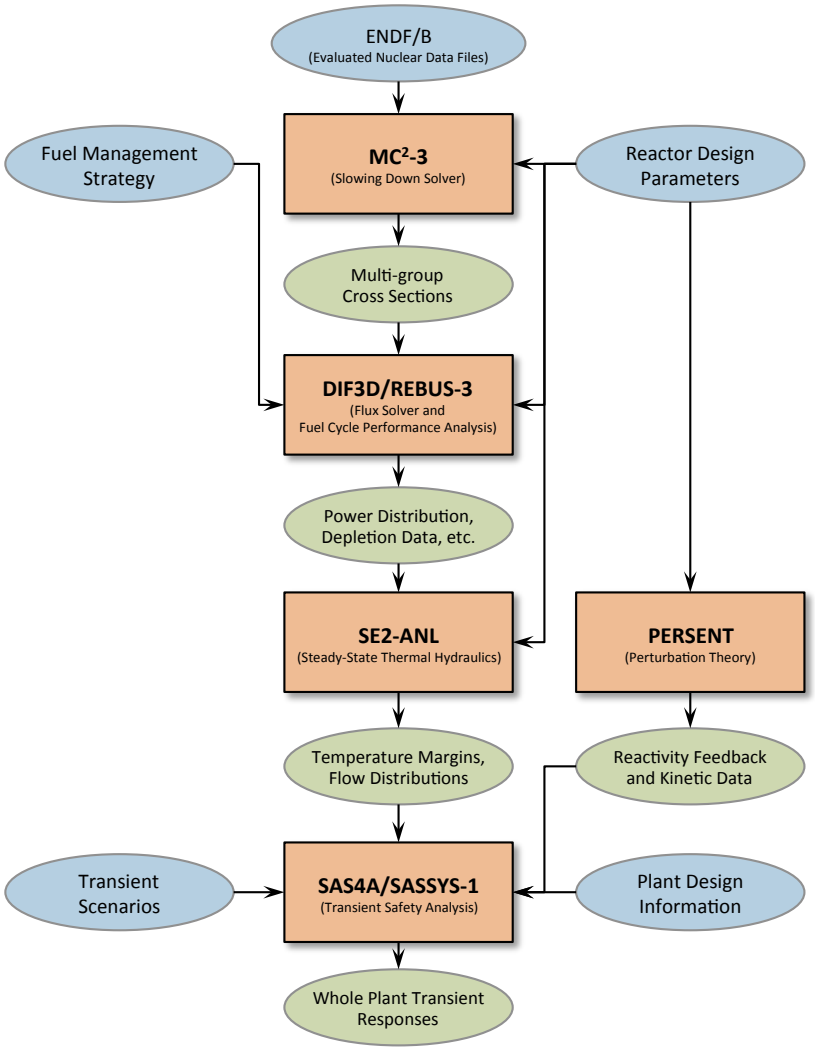
The Safety Licensing Research Plan developed a fairly comprehensive spectrum of the safety analyses expected to be important in an authorization basis or license application. The range of relevant events and accident scenarios was selected based on three risk categories: anticipated operational occurrences (AOO), design-basis accidents (DBA), and beyond design-basis accidents (BDBA). Within each risk category, experts defined event tables to identify the relevant phenomena that analysis codes need to model. Using the detailed tables published in Reference [2], high-level functional areas have been defined and are presented in Table 2. The event tables also included domestic and international computer codes that might be used to perform the associated analysis. Domestic computer codes associated with each functional area are also listed in Table 2.

Table 2: Functional Areas for Fast Reactor Design and Safety Analysis

Functional Area	Code(s)	Role
Steady State Characterization		
Neutron and Gamma Transport	MC^2-3	Primary
	DIF3D	Primary
Fuel Cycle Performance	REBUS	Primary
	ORIGEN	Secondary
Fuel Performance	LIFE-METAL	Primary
	SAS4A/SASSYS-1	Secondary
Core-Wide Thermal Hydraulics	SAS4A/SASSYS-1	Primary
	SUPERENERGY	Primary
Transient Analyses		
Fission Gas Behavior	SAS4A/SASSYS-1	Primary
	LIFE-METAL	Secondary
Fuel and Clad Motion	SAS4A/SASSYS-1	Primary
	LIFE-METAL	Secondary
Primary/Intermediate System Heat Transport	SAS4A/SASSYS-1	Primary
Structural Response	NUBOW-3D	Primary
	SAS4A/SASSYS-1	Primary
Inherent Reactivity Feedback	PERSENT	Primary
	SAS4A/SASSYS-1	Primary
Passive Heat Removal	SAS4A/SASSYS-1	Primary
Sodium-Water Interactions	SWAAM-II	Primary
Sodium Fires	MELCOR	Primary
	CONTAIN-LMR	Secondary
	NACOM	Tertiary
	SOFIRE-II	Tertiary
Source Term	ORIGEN	Primary
	MELCOR	Primary
	CONTAIN-LMR	Secondary

3.3 Design and Safety Analysis Workflow

The vast majority of events and accident scenarios defined in Reference [2] rely on the same sequence of computer codes. This sequence, or workflow, is shown in Figure 1. The codes used in this workflow are summarized in Table 3. An assessment of the status of each of these codes is given in Section 5, below.



[1]

Figure 1: Fast Reactor Design and Safety Analysis Workflow

Table 3: Summary Description of Critical Fast Reactor Design and Safety Analysis Methods

Code	Version	Description
MC^2	3.0	Software to prepare multi-group cross section input for the homogenized reactor analysis methodology.
DIF3D/VARIANT	11.0	Steady-state solvers of the neutral particle diffusion or transport equation on structured grids for conventional homogenized reactor modeling
REBUS	11.0	A reactor fuel cycle analysis capability designed to model fast spectrum reactor operation in a commercial environment with both breeder (fast) and burner (thermal) reactors.
GAMSOR	11.0	Software to orchestrate the steady-state solve of the coupled neutron and gamma transport equations.
SUPERENERGY	11.0	A steady-state thermal hydraulics software designed to treat ducted, wire-wrapped pin fuel assemblies typical of sodium cooled fast spectrum reactors.
PERSENT	11.0	A perturbation and sensitivity code wrapped around the DIF3D code for use in generating point kinetics related reactivity coefficients and assessing uncertainties due to cross section measurement errors.
SAS4A/SASSYS-1 [25]	5.2.2	Simulation tool used to perform deterministic analysis of anticipated events as well as design basis and beyond design basis accidents for fast reactors.

3.4 Limitations on Scope

The previous section defines the essential analysis codes required for nearly all events and accident scenarios. However, comprehensive design and safety analysis requires additional analysis methods beyond those evaluated for this report. Other analysis methods may include codes sponsored by the U.S. Nuclear Regulatory Commission, maintained by private industry, available as commercial software, or developed by foreign organizations. Overall, the SNL report identified over 60 computer codes that are currently available in the international community to perform various aspects of safety analysis.

Non-commercial domestic codes identified in the SNL report, but not included in Table 3, are summarized in Table 4 along with the rationale for their omission from the detailed gap analysis presented in Section 5.

Table 4: Additional Codes Supporting Fast Reactor Analysis

Code	Purpose / Rationale for Exclusion
CONTAIN-LMR	Containment analysis for *liquid metal-cooled reactors* (LMR). No longer officially maintained. The original CONTAIN code is no longer supported by the U.S. NRC. The unique features of CONTAIN-LMR were migrated into the CONTAIN-2 source tree and have subsequently been merged into MELCOR. [24] Although this migration was supported by DOE, MELCOR is maintained under the direction of the U.S. NRC. The updates to MELCOR also include some of the sodium spray and pool fire modeling capabilities from NACOM and SOFIRE. [21] [22] [23] [24]
LIFE-METAL	Thermal and mechanical fuel performance for metallic fuel. LIFE-METAL [26] is an extension of LIFE-4 (for oxide fuel [27]) that includes key phenomena applicable to metallic fuel and metallic fuel properties. It continues to play an important role in fuel performance modeling. For the purposes of safety analysis, basic metallic fuel property correlations and models required for transient accident conditions (fuel constituent redistribution, fuel failure, melting, and relocation) are being developed in SAS4A/SASSYS-1.
MACCS	Offsite consequence analysis. Maintained under the direction of the U.S. NRC. In 2004, the U.S. DOE evaluated MACCS against DOE safety software quality assurance criteria and subsequently listed it in DOE's safety software Central Registry as a safety analysis toolbox code. [28]
MCNP	General purpose Monte Carlo transport code. [29] Maintained under direction of the U.S. National Nuclear Security Administration. Although MCNP is often used to produce reference core solutions, it is not appropriate for use in transient safety analyses. MCNP is very useful for performing design-basis activation and shielding analyses.
MELCOR	LWR severe accident framework. [30] Maintained under the direction of the U.S. NRC. At the time of the SNL report, MELCOR did not contain fast reactor specific capabilities. Since then, capabilities from CONTAIN-LMR, NACOM, and SOFIRE-II have been incorporated into MELCOR.
MELTSPREAD	Analysis of transient spreading of core debris melts. [31] Although identified in the SNL report, MELTSPREAD targets the spreading behavior of high temperature melts flowing over concrete and/or steel surfaces in air or water. Therefore, it is not directly applicable to LMRs, but its modeling capabilities may be relevant for sodium pool spreading.

Code	Purpose / Rationale for Exclusion
MELT-III	Initiating-phase analysis for hypothetical core disruption accidents. [32] [33] No longer maintained. The most recent public report was published in 1974. An unpublished document indicates the last update was made in 1978. The capabilities of MELT-III, which focused on FFTF analyses, have been superseded by the capabilities in SAS4A/SASSYS-1.
NACOM	Sodium spray fire analysis. [34] Not currently maintained. The most recent available report was published in 1980 by Brookhaven National Laboratory as NUREG/CR-1405. Original source code was recently recovered, compiled, and tested at Argonne National Laboratory, however no further developments have been made. MELCOR includes the sodium spray fire models from NACOM to support containment analyses, but separate analysis capabilities will be needed for sodium facilities.
NUBOW-3D	Core-wide structural deformation and restraint system analysis. [35] [36] Although NUBOW is not actively being developed, it is being used to support the PGSFR program and was used by Purdue University to assess reactivity feedback effects due to core deformation. Advanced structural analysis codes such as ANSYS may be an alternative for NUBOW, however developments would be needed to include the effects of irradiation creep and swelling due to thermal and fast flux exposure. Due to the importance of core radial expansion effects during transients, continued development of NUBOW may be a future priority.
ORIGEN-2	Isotope generation and depletion code. [37] [38] ORIGEN-2 as a stand-alone code is no longer maintained, with the final update provided in 2002. The current version of ORIGEN is integrated into the SCALE software package maintained under the direction of the U.S. NRC. Most fast reactor fuel cycle analyses require the capabilities of REBUS, however ORIGEN is a better choice when evaluating detailed source-term compositions or long-term spent fuel storage conditions.
RELAP5	Light-water reactor transient analyses. Not generally applicable for fast reactor transient analyses. Two version of RELAP5 are commonly known: RELAP5/MOD 3.3 [39] and RELAP5-3D. RELAP5/MOD 3.3 is maintained by the U.S. NRC, but it is being phased out in favor of TRACE. [40] Although RELAP5-3D can support non-water coolants, several features important for fast reactor safety analyses are not present, including metallic fuel property correlations, pre-transient fuel irradiation effects, complex fast spectrum reactivity feedback effects, relevant fuel failure mechanisms, in-pin and ex-pin fuel relocation models, and sodium boiling models.

Code	Purpose / Rationale for Exclusion
SIMMER	Transition-phase core disruption analysis. [41] No longer maintained in the US. The original SIMMER code (up to SIMMER-II) was developed by Los Alamos National Laboratory until about 1990. Around that time, the code was transferred to the Power Reactor and Nuclear Development Corporation [subsequently the Japan Nuclear Cycle Development Institute (JNC), Commissariat à l'énergie Atomique (CEA), and Forschungszentrum Karlsruhe (FZK)] for continued development as SIMMER-III.
SOFIRE-II	Transient analysis of sodium pool fires. [42] Not currently maintained. SOFIRE-II was developed by the Atomics International Division of Rockwell in 1973. Original source code was recently recovered, compiled, and tested at Argonne National Laboratory, however no further developments have been made in the U.S. A few variants of the software have been implemented by international organizations. MELCOR includes the sodium pool fire models from SOFIRE-II to support containment analyses, but separate pool file modeling capabilities will be needed for sodium facilities.
SUPERENERGY-2	Sub-channel thermal-hydraulics analysis. [43] No longer maintained. The most recent available report is from 1980. Argonne's version of SUPERENERGY (Table 3) is a significantly modified version of SUPERENERGY-2 that adds coupling to the heating calculation methods in the DIF3D code system. In addition, models were added for hot spot analysis, fuel element temperature calculations, and allocation of coolant flow orifices subject to thermal performance criteria.
SWAAM-II	Steam generator sodium-water interaction analysis. [44] Limited maintenance at Argonne. Priorities for the validation and use of SWAAM-II will depend on analysis needs for SFR concepts with a steam power-conversion system.
VARI3D	Neutronics perturbation and sensitivity analysis. [45] [46] VARI3D has been deprecated. The Nuclear Energy Advanced Modeling and Simulations (NEAMS) program sponsored the development of PERSENT (see Table 3), which is a transport-theory based replacement for VARI3D.
VENUS-II	Prompt criticality core disassembly analysis. [47] No longer maintained. Information on the last update was published in 1972.

3.5 Additional Considerations

Verification and validation (V&V) processes are important elements of software development and use. However, there are at least two different perspectives. The IEEE defines standards for software V&V with respect to the developer to ensure that software development conforms to the defined requirements and that the product satisfies its intended use. [48] This form of V&V is expected to be a part of the software quality assurance goals summarized in Section 4.1.

From the perspective of the user, software V&V relates to the applicability of a given computer code to a particular *design* and includes input verification and model validation. The computer codes documented in Section 5 are intended to be generally applicable to a wide range of fast spectrum systems, but it is not practical for the developers to anticipate all conceivable design configurations. End users are responsible for determining the applicable regulatory requirements, identifying the specific reactor design, assessing selected codes with respect to relevant separate effects or integral testing results, verifying computer code input and selected modeling options, and quantifying uncertainties in the applied models. This form of V&V is *not* addressed in the current document.

4 Goals

As noted above, U.S. Government policy changes in 1994 abruptly canceled nearly all fast reactor research in the United States. In the subsequent two decades, very few resources were dedicated to preserving the design and safety analysis capabilities that had been established during the preceding four decades. The high-level objective of this work is to document the gaps that exist in the *development*, *maintenance*, and *use* of key fast reactor analysis methods (computer codes) that are needed to support an authorization basis or commercial licensing for deployment of fast reactor technology.

Seven goals have been identified to help organize the gaps analysis (see Table 5). At a high-level, the Safety Licensing Research Plan identified knowledge preservation as an overarching gap. Five of the seven goals described in this section represent key attributes of software development that are required to preserve and maintain fast reactor analysis capabilities. The full collection of codes documented in Section 5 are maintained by only a few stewards. Should they retire before knowledge has been transferred to the next generation of developers, DOE's capability to defend fast reactor safety authorization and licensing will become considerably more expensive and may even be lost. One of the remaining goals is to identify and address critical modeling gaps through additional development. The last — and perhaps most critical — goal is to establish basic user support so that a knowledgeable user community with expertise unique to fast reactor analyses can be expanded.

Table 5: Goals for Fast Reactor Methods Development

Maintenance
•Software Quality Assurance
•Documentation
•Configuration Management
•Code Standards
•Compatibility
Development
•Modeling Gaps
Use
•User Support

4.1 Implement Active Software Quality Assurance Program

A key goal for long-term sustainable and traceable software development is to establish, implement, and maintain an active SQA program. An SQA *plan* is the top-level document that clearly defines project scope, safety classification and grading, roles and responsibilities, standards and conventions, and various software work activities that are essential for code maintenance. For example, see Reference [8].

The development of modern software quality assurance (SQA) practices post-dates the cancellation of fast reactor research in 1994. Prior to that time, the development of fast reactor analysis methods generally involved teams of experts and dedicated code managers that worked in concert with users who applied the methods to operating facilities. Implementation of an active SQA program helps ensure continuity and traceability of development activities. It also provides an interface for code qualification and regulatory acceptance activities.

4.2 Document Code Theory, Implementation, and Use

Documentation plays two significant roles. It describes the theory and software implementation (e.g. equations, assumptions, discretization, algorithms) for developers, and it describes the theory and usage (e.g. intended scope, user interfaces, tutorials) for users. Software documentation can be written as separate text or embedded in the source code as comments for developers.

Unfortunately, rapid software development cycles and limited resource allocations tend to favor the expansion of code features over comprehensive documentation (or even code stability). The goal is to synchronize both technical and user documentation with software development activities.

4.3 Establish Software Configuration Management

Software configuration management involves the tracking and control of software (and documentation) changes in order to maintain version control and to define baselines. One of the earliest forms of configuration management was the Source Code Control System (SCCS) introduced in 1972. [49] The Revision Control System (RCS) was introduced in 1982 [50] and the Concurrent Versioning System (CVS) in 1986. At present, there are approximately two-dozen version control systems available.

Some, but not all, of the fast reactor analysis codes were historically maintained under SCCS. In some cases, the software was maintained without formal revision control measures, other than the prevailing QA practices, and were preserved by source code managers as file and directory archives and patch files.* Due to the dramatic evolution of computing platforms and data storage over the last thirty years, most of this history has been lost. Computing platforms advanced from mainframes, to workstations, to personal computers. Mass data storage advanced from tape archives, to optical media, to hard drives, and now massive cloud-based storage. Archived records were not always copied to the newest storage medium.

More recently, the codes documented in Section 5 have been preserved in Subversion, [51] a server-based version control system that was first released in 2004. Beyond tracking source changes, however, the goals of software configuration management are to perform unit testing, regression testing, release management, and issue tracking, along with other software maintenance activities. Records of these activities need to be maintained during the lifetime of the software.

4.4 Ensure Compliance with Modern Code Standards

Compiled programming languages are often used when high computational performance is needed. The first high-level programming language, Fortran, was initially released in 1957. At the time, it was unique to a particular machine (the IBM 704). Other computer manufacturers implemented versions of Fortran

* It is interesting to note that even the Linux kernel was maintained in this manner until 2002. (https://git-scm.com/book/en/v2/Getting-Started-A-Short-History-of-Git)

for their own machines, but there was no guarantee that a program written for one type of machine would work on another. By 1962, a committee was formed to pursue the goal of standardization, and in 1966 Fortran became the first programming language to be defined by a formal standard. [52]

Since then, computer scientists have developed numerous additional languages, and international standards organizations have continued the effort to define formal standards. However, language standards are not static; they evolve as software development paradigms change. The latest standard for Fortran 2008 was adopted in 2010, [53] and the next revision is expected later this year (2018). Likewise, the latest standards for C and C++ are from 2011 and 2017, respectively. [54] [55]

It is *not* a goal to recast or rewrite all fast reactor analysis codes using the newest standards. In fact, compiler vendors often take years to implement new language specifications. It *is* the goal to maintain pace with the widespread adoption of standards and the deprecation of obsolete language elements as code updates are implemented. Modern compilers support sophisticated static and dynamic code analysis capabilities that can help reduce errors. This goal ensures that those capabilities can be used and that the analysis codes will remain viable on future hardware platforms.

4.5 Maintain Cross-Platform Compatibility

It is ironic that the end of methods development support in 1994 coincided with a radical shift in the computing hardware used for design and safety analysis activities. Throughout the 1980s, most analysis was done on large mainframe computers. With the adoption of workstation-class computing resources tailored for scientific computing, analysis codes were ported to new platforms and operating systems in the early 1990s. At the time, workstations offered significant performance advantages over personal computers. The increasing performance of personal computers led to another shift in the early 2000s towards commodity hardware dominated by a single chip architecture. Even with common hardware and well-defined language specifications, there are different operating systems that present challenges for maintaining and verifying build, test, and run-time environments.

Operating systems and hardware architectures will continue to evolve. The goal is to maintain compatibility with a broad, but reasonable, subset of available platforms. Cross-platform support helps improve code quality, facilitates adoption of changing software and hardware architectures, and expands the user base to a larger audience.

4.6 Identify and Resolve Critical Modeling Gaps

Despite extensive development and maturity, methods development support ended abruptly and critical modeling gaps exist. Some of these modeling gaps are the result of incomplete implementation of key features. For example, the integration of metallic fuel failure models into SAS4A/SASSYS-1 was never completed. Other gaps include outdated assumptions that no longer apply to modern computing platforms, such as limitations on memory utilization, parallel scalability, and big data management. Still other gaps are the result of evolving needs, such as uncertainty quantification and performance optimization, for supporting an authorization basis or license application for a fast reactor.

The objective of this goal is to identify and prioritize the critical modeling gaps that exist in each of the analysis codes reviewed in Section 5. Critical modeling gaps are limited to those essential for supporting near-term fast reactor deployment. Prioritization depends on the projected time-frame for gap closure and the potential impact on supporting an authorization basis or license application.

4.7 Provide User Support

The Safety Licensing Research Plan identified “knowledge preservation and management as the most pressing need for the SFR community.” Within the Codes and Methods topic, the key recommendation to resolve this need “is to ensure that the codes which are needed to support a safety case have the

appropriate level of stewardship and user-base within the DOE labs and academia." Goals for achieving the appropriate level of *stewardship* have been defined in the preceding sub-sections. Developing a robust *user-base* requires a sustained, domestic interest in fast reactor technology.

Expertise in the application of fast reactor analysis methods and interpretation of simulation outcomes is limited to an extremely small population. In the academic environment, very few students are exposed to fast reactor technology, design, or analysis concepts. As a result, support in the *use* of fast reactor analysis methods is usually *ad-hoc*. With industry, support is sometimes more formal through contractual agreement, however it takes place only between the two parties and does not generally benefit the wider community.

Interactions with industry have revealed that there is a strong need for and interest in broader user support. Support for a user-base needs to include formal issue reporting mechanisms, training opportunities, collaboration resources, and user-group meetings.

5 Fast Reactor Code Status

5.1 MC² Version 3

Version 3 of the MC^2 code was built as an integral component of a fast spectrum reactor analysis capability under the NEAMS program of DOE. [56] [57] [58] [59] Its primary function is to generate multi-group cross sections usable in steady state, fuel cycle, and transient reactor analysis efforts. Compared to the previous version, MC^2-2, [60] the new version greatly improves the resonance self-shielding and spectrum calculation methodology and has a built in one-dimensional lattice cell calculation capability. MC^2 solves the consistent P1 multi-group transport equation for the fundamental mode spectra in ultrafine (~2000) or hyperfine (~400,000) group levels given processed neutron cross section data taken from ENDF/B or equivalent data files. The ENDF/B data files are processed and prepared for use by others by MC^2 expert developers at Argonne National Laboratory.

Because fast spectrum systems have large mean free paths, MC^2 is primarily used on infinite homogeneous medium calculations for homogenized assembly compositions (i.e. the repeating structure of the reactor geometry). With its built in one-dimensional modeling capability, the user can alternatively use MC^2 on a heterogeneous slab or cylindrical unit cell problem on the ultrafine or hyperfine group levels. In the resolved resonance range, pointwise cross sections are reconstructed with Doppler broadening at the user specified isotopic temperatures. The pointwise cross sections are directly used in the hyperfine group calculation (~400,000 groups) whereas for the ultrafine group (~2000 groups) calculation, self-shielded cross sections are prepared by numerical integration of the pointwise cross sections using the narrow resonance approximation. For both the hyperfine and ultrafine group calculations, unresolved resonances are self-shielded using the analytic resonance integral method.

Using an external flux solving capability, the MC^2 software can be applied to full core problems to better capture the spectral interaction between core and reflector regions. While typically done in homogenized R-Z geometries, the functionality is only limited by the computational capability available to execute the full core problem.

5.1.1 Software Quality Assurance

There is no formal software control guidance document for MC^2. There is no written procedure for how the software is to be updated and distributed or how the reported software errors are to be addressed. Its quality and accuracy are purely driven by the passion of the two main developers and what limited funding is provided by the NEAMS project. Given there are only two developers, they act solely and independently when it comes to software updates and maintenance, with no independent checking by other staff members. Software quality assurance is therefore a by-product of the need to use of the software to support various independent missions.

Beyond the control procedures, software quality assurance of codes that work with continuous energy cross section data is one of the more difficult aspects of modern neutronics software development. Many fuel isotopes have thousands of resolved resonances each of which has significant measurement errors and, when combined with the uncertainties in the unresolved resonance range, make the entire cross section evaluation quite prone to significant error hence the historical presence of multiple evaluations. Verification becomes even more complex due to the fact that we cannot construct infinite sized reactor problems and must study problems with significant amounts of leakage or material heterogeneities which pose problems that are beyond the computational capabilities of MC^2. While we can construct trivial checks for the base algorithms embedded in the software, testing out the processing of all isotopes for a given evaluated cross section data set is still not practical today and relies upon tedious code to code comparisons between MC^2 and continuous energy treatments like that in MCNP.

The existing MC^2 software testing was created for checking the software use when deployed outside of Argonne National Laboratory. After successfully compiling the software on the target machine, the user can execute 19 test problems, commonly referred to as top-down, to verify that the software is trustable. The test suite consists of 17 test problems for which the multi-group cross section data and k_{eff} (steady state eigenvalue) were compared against MCNP calculations of the same geometry by the developers. The reference outputs of each is provided and used as the comparison point. In each calculation, the capture and fission related cross sections were tallied in MCNP and an accurate eigenvalue is relied upon to assess the accuracy of the remaining cross sections (scattering). There are two additional verification test problems included with the MC^2 distribution that test out the utility operations of MC^2 such as merging product cross section files.

For additional accuracy, the developers maintain ~30 test problems that use the RZ transport option to assess the bulk cross section accuracy by eigenvalue comparisons. These test problems are derived from physical experiments but only representative of those experiments. In that manner, they amount to additional comparisons against MCNP calculations of larger scale problems for a given cross section data library. There is no current documentation indicating how frequently those 30 test problems are executed with regard to any software distribution or maintenance task.

5.1.2 Code Documentation

Every MC^2 release contains a manual and a user installation guide. The manual covers all details of the code inputs, outputs, and methodology behind using MC^2 for generating multi-group cross section data. The (ENDF) processed data files provided by ANL are discussed and the input and output of several example problems are discussed. A final section is dedicated to the programming structure of the software. It has been updated as new features are added, and distributed as appropriate, but the updates have not gone through the formal report acceptance and publication procedure at Argonne National Laboratory.

The installation guide covers the organization of the software distribution, the code interface, and discusses how to use the software in Windows or Linux. The MC^2 software poses a complete break in the logic of execution with MC^2 and thus the guide informs the user of how to execute the software and what outputs to expect from it. It guides the user on how to compile the software, convert the ASCII libraries to binary, and run various scripts which perform the software verification.

5.1.3 Configuration Management

The MC^2 software is maintained in an export controlled repository at Argonne National Laboratory. Because MC^2 is a collaborative effort between Argonne National Laboratory and the University of Michigan, the software has two simultaneous development paths and is periodically merged by the Argonne developer. This is obviously a tedious process and requires considerable effort to make a consistent code that covers the reported features from all publications. Fortunately, a bulk of the software

is not modified by the two developers such that these merges only require heavy modification to the input and output components of the software.

While the repository is exposed to the continuous regression testing, the software in the repository rarely represents the current deployed executable and thus is not part of the continuous regression testing. In fact, the MC^2 software updates to the repository are normally only done when a new version of the software is to be released. This behavior is closely linked to the difficulty in merging the branches and the limited funding to do any of the actual work described in this section.

5.1.4 Code Standards

The entire MC^2 software was written in a Fortran 95 compatible form. However, this does not mean the software follows modern practices of data encapsulation. Most of the Fortran in MC^2 would be better described as Fortran 77 with the elimination of labeled statements in favor of modern Fortran alternatives. This is consistent with converting its predecessor from a mixture of Fortran 77 and Fortran 66 to Fortran 90+ formatting.

With regard to problem areas, there is heavy usage of passed pointers into subroutines for allocation of arrays and thus unnecessary abstraction of allocated arrays that would be better performed, readable, and testable as encapsulated allocations. Further, the arrays are all collected as globally accessible components across several Fortran modules which is indicative of a Fortran 77 organization with common blocks rather than a Fortran 90 one. This is also expected given its origin of MC^2 as a mixed Fortran 66 and Fortran 77 code. Finally, the input structure of the coding relies upon the Fortran namelist scheme for input which leaves little help for the user when there are input mistakes as opposed to the original Argonne Reactor Code package concept of helping the user fix their mistakes with clear messages. None of these problem areas prevent the software from being used reliably today and act more like a barrier to improving the functionality of the code going forward.

5.1.5 Compatibility

Because there are no deprecated Fortran coding features or data parallelization methodologies used in MC^2, it is applicable to most modern computing platforms. The common problem with data file portability is eliminated by providing asci to binary convertor routines. At present, the MC^2 compiler supports the intel and gnu compilers on Windows, Linux, and macOS and has proven execution on all three platforms.

5.1.6 Critical Modeling Gaps

For fast spectrum reactors, there are no critical modeling gaps in the MC^2 software. There is a major cross-section processing code step called ETOE which performs energy-to-energy transformations for nuclear data, and this needs considerable improvement. A collection of other improvement areas for MC^2 are recommended to reduce unnecessary errors. The major areas to work on in MC^2 are

1) Processing of ENDF data to MC^2 libraries
2) Gamma Library: The handling of gamma cross section data
3) Data Treatments: Missing (N,3N) and delay gamma data treatments
4) Complex Geometry: Difficult to use for complex heterogeneous structures

To allow for rapid evaluation of Doppler broadening, MC^2 relies upon a processing code to process the resonance data from ENDF and put it into a resonance form for MC^2. It was written primarily in Fortran 66 (it was written before 1976) but is mostly Fortran 77 standard today with heavy modifications over the years. The main problem with this processing code is that with each new ENDF evaluation, the specific resonance treatments for isotopes are changed and in some revisions, new resonance representations have

been added. Beyond the obvious lack of being able to treat a new resonance representation, interpolation errors in the transformation of resonance data from the ENDF format to that used by MC^2 occur frequently. There is no manual or real theory document to work from for this processing code and we presently rely upon the expert developers to maintain that knowledge and carry out the conversion for each ENDF revision. To identify the problems with the interpolation requires detailed, tedious comparisons to a code that can carry out continuous energy calculations such as MCNP which we again rely upon the expert developers to perform. At a minimum, the documentation needs to be updated to describe the processing code and its usage.

For the gamma library gap, a necessary functional component for investigating accuracy is missing. This problem is partly related to the historical treatment of gamma transport in the ARC system [61] as it was added as an afterthought more than an integral component. The MC^2 specific issue on this topic is the inability of the software to provide an arbitrary energy group gamma library. For neutron cross sections, one can select an energy structure that is a child energy structure of the ultrafine structure. This allows one to investigate the importance of refinements in the energy structure for any problem by refining the group structure. For gamma ray library generation, the user can only utilize a 21 group library.

The gamma producing neutron capture cross sections are self-shielded in MC^2 to get the gamma production. However, this energy self-shielding is only applied on the compositional level and it is not presently possible to use the external RZ neutron flux calculation or internal 1D geometry calculation. As evident from the fixed structure, there are no energy details for the gamma cross sections in MC^2 which is the minimal requirement to correct the gap. It is important to note that the gamma power production in reactors only amounts to a few percent of the total power and thus the errors introduced by the preceding approximations are rather minimal to the accuracy of the overall calculations.

In the Data Treatment gap, the accuracy of some reactor problems can be impacted significantly due to the lack of treatment of the stated data. The missing (N,3N) treatment is particularly important for higher energy spectrum (compact metal fueled reactors as opposed to oxide fueled ones). While the current methodology does keep the neutron balance such that DIF3D will predict the proper eigenvalue, the fast flux distribution has been observed to vary significantly from MCNP due to the (N,3N) treatment. The missing gamma data is a follow on to the first gap where delay gamma data is not properly accounted for in the library generation. Because of the spectral differences in prompt and delay gamma data, this mistake alters the accuracy of gamma ray energy deposition in the non-fueled regions of a reactor core. As one should infer, these errors are again minor and expected to be less than 5% on the existing predictions with MC^2.

The complex geometry limitation is a reoccurring problem with generating cross section data whether it be fast or thermal spectrum reactors. In general, for fast spectrum reactors, the geometrical heterogeneities within a fuel assembly (10-20 cm) are not important as the mean free path of neutrons (~15 cm) is on the order of the assembly size. It is known from the ZPR and ZPPR experiments that the separation of fertile fuel (U-238) from fissile fuel (U-235) by coolant and structural materials causes substantial heterogeneity effects on a ~5 cm geometrical basis which must be treated carefully in MC^2 to produce accurate region averaged cross sections.

For demonstration or production fast reactor plants, the use of thick control rods is one known heterogeneity that must be handled properly. In this case, the explicit control rods act as a near black absorbers but when applied homogeneously over an assembly their effective absorption is altered as they are equally mixed with coolant and structure. The same goes for other types of spatial heterogeneities such as blanket pins, target pins, or even blanket assemblies inserted internally to the reactor core as the local gradient associated with a homogeneous assembly approximation versus a heterogeneous one can be considerable. From past experience, errors in spatial heterogeneities have been demonstrated upwards of 200 pcm in the eigenvalue which has minor impacts on the control rod insertion and or fuel enrichment. In the case of blanket pins, the pin reaction rate can be in error up to 5%.

5.1.7 User Support

The user support for this software is currently unfunded and provided as needed. If the remaining primary developer chooses to leave the laboratory, it is not clear who would functionally provide user support on the software. Given there is some development money still being focused on the software from the NEAMS program, it should be clear that the stated gaps are being dealt with although at a much reduced pace with regard to production releasable software development. Overall, the user support is prompt and good given the modest user base of the software.

5.2 DIF3D/VARIANT

DIF3D was originally built as a three-dimensional solver of the multi-group diffusion equation for structured grid geometries. [62] The diffusion equation, with proper cross section data, has proven to be a very reliable means to predict the fuel cycle behavior of thermal and fast spectrum nuclear reactors. [63] The DIF3D code was initially built in the late 1970s using a finite difference spatial differencing methodology applied to the diffusion equation referred to as DIF3D-FD. Later, in the early 1980s, a transverse integrated nodal methodology, referred to as DIF3D-Nodal, was built into DIF3D to improve the performance on large scale reactor problems. [64] [65] Finally, with a research design goal of non-proliferation fast spectrum reactors that eliminate blanket assemblies, the DIF3D-VARIANT solver was added to DIF3D which extends the concept of DIF3D-Nodal to a functional three-dimensional transport code based upon spherical harmonic expansions in angle. [66] [67] [68] [69] [70] [71]

Over its 40 year history, DIF3D has been applied to numerous fast and thermal spectrum reactor analysis projects. For those problems with experimental measurements, DIF3D performed exceptionally well yielding results of consistent accuracy to those produced by MCNP at a fraction of the cost. With all of these successes, it is important to point out that DIF3D does have limitations for general purpose usage such as the serpentine geometry of the Advanced Test Reactor and other geometrically non-regular systems.

5.2.1 Software Quality Assurance

There is no formal software control guidance document for DIF3D. There is no written procedure for how the software is to be updated and distributed or how the reported software errors are to be addressed. Its quality and accuracy are a byproduct of the amount of free time available by the one active maintainer of the software. All software updates are applied by the current maintainer with no checking by any other staff member.

Given the age of the software, there are no component or algorithm tests of the DIF3D software, only top-down tests. At present there are 34 test problems in the test suite which reference outputs are provided along with a python based checking script that is set to detect unacceptable compiler/platform errors derived from compilation problems. With funding provided by the NEAMS program, the software was moved from a SCCS repository on the deprecated SUN workstation system to a Subversion repository [51] in ~2008. In addition, the incorporation of multiple modifications from several developers were collected and merged into the main branch to create DIF3D 10.0 and the latest version 11.0 from the previous DIF3D 8.0. (DIF3D 9.0 was never formally released but its changes were also included). This work produced a new manual that included almost a decade of updates to DIF3D. During that work several new test problems were created and added to the software distribution test suite extending it from ~15 test problems to the current 34. These test problems were added as the existing suite was found to be deficient at verifying the existing options most often used in the DIF3D software.

The DIF3D code solves a deterministic diffusion or transport equation and thus can be directly compared to analytic solutions of the diffusion or transport equation. While this type of work was done in the past, it is poorly documented and not part of the current testing apparatus today. Further, one can use various Monte Carlo codes in multi-group mode to obtain a solution to the forward and adjoint transport equation

without concern for space-angle discretization thereby providing the best code-to-code comparison capability. Again, while this technique was used in the past, it is poorly documented and not part of the testing apparatus today. Beyond the verification test suite, the current developers maintain a few validation cases where DIF3D is used with a fixed cross section data and input to produce results for a measured experimental results. The frequency of using these test cases has not been documented nor are the test cases themselves documented.

5.2.2 Code Documentation

The original DIF3D-FD manual was written in 1978 and has not been updated since. The DIF3D-Nodal manual was written in 1983 and has not be updated since. The DIF3D-VARIANT manual was written in 1995 and was heavily updated in 2013 to account for the numerous source code changes made to it and DIF3D in the previous decade. [72] The DIF3D-FD and DIF3D-Nodal software methodology has barely been modified since the original documents and thus the manuals are still a sufficient description of the algorithms being used in the DIF3D software although there are known errors with the equations and descriptions in the manuals, inconsistencies in the actual implementations in the code, and numerous undocumented bug fixes.

There is no adequate document describing the execution process of the DIF3D software and users are only given a simple README file to understand how to use it in the distribution. There is no developers guide to provide assistance on the infrastructure design of the DIF3D software or software execution layout. Thus beyond the few developers that already know how the software operates, extending the software beyond its current input and output usage is impractical for most potential developers or collaborators. There are numerous features described in the manuals that have been deprecated over time due to platform specific issues none of which have been documented.

The input related errors caused by bad user inputs are generally well handled by DIF3D. They give guidance for the user to identify the problem section of input and how to correct it. In some cases, however, the error handler of DIF3D produces no useful messages and thus the expert user or developer must be relied upon to give guidance as to the specific problem with the user input. Some output produced by DIF3D is also not well explained. In some cases, the output file directs the user to read the manual instead of just producing a clear concise statement as to what the actually output means. In the case of DIF3D-VARIANT, an existing output from the code is wrong and not described in any manual. Conversely, an output from DIF3D-Nodal is correct, but the actual description in the manual is wrong. These types of problems are not expansive, but they exist and experienced users know how to avoid them.

5.2.3 Configuration Management

The existing DIF3D software is not export controlled, but it is housed in an export controlled Subversion repository as it is linked with other exported controlled software. Its inclusion into the Subversion repository was a considerable modification of the source code to allow it to transition from a multi-step c-shell script driven maintenance system in SCCS on the deprecated Sun workstation system to a modern Linux-based system with python based scripting. The Buildbot [73] service was setup to carry out nightly regression tests of the DIF3D software since 2008 where more rigorous checking procedures were adopted in 2009-2010.

The Buildbot system checks out the latest revision of the software, compiles it and all associated utilities, runs all of the top-down verification tests, and does a number by number, line by line comparison with reference outputs. The reference outputs are stored in the Subversion repository and any updates to those outputs have been documented in the repository since 2008. The outputs were manually verified to be accurate by a qualified software developer although the test problems rarely have analytic solutions or any type of documented code-to-code comparisons. The results of every nightly Buildbot comparison are

maintained for a period of 5 years and disposed of after that. The Buildbot service runs on a dedicated Linux workstation paid for via the NEAMS program.

A number tolerance is used by Buildbot on every checked number between the output and reference output. This tolerance is set based upon the ability of the python script to identify the type of output table being assessed and the individual column associated with each number in the table. The tolerance is thus tailored to the inherent accuracy of the number being printed by the code (i.e. flux, power, volume, eigenvalue, etc.). The selection of the tolerance and identification of each table is not documented but was chosen by an expert developer of the software package. The verification itself is checked by having the python testing script check a fabricated test output against an associated reference output file. In the fabricated test output, multiple lines of a given table are duplicated and each number in each column of each table is manually altered to values above and below the tolerance. The python testing routine reports errors when the tolerance is not met and the output was manually verified to ensure that the testing python script properly flags those modifications that fail the tolerance check. The output from the checking script is stored as the true reference of the checking script itself and is used as verification of the verification script every time the software checking procedure is executed. The entire python testing apparatus is part of the software distribution and setup such that any person with the source code sees the repository exactly the way that it is maintained and execute the verification exactly the way the Buildbot software does it. While not perfect, this approach has minimized the potential for introducing software errors into the distribution.

5.2.4 Code Standards

The DIF3D software is old and thus contains a mixture of Fortran 66, Fortran 77, Fortran 90, and Fortran 95. The more modern Fortran additions were a result of modifying the DIF3D-VARIANT software to extend its capabilities, run faster, and fix mistakes in its original implementation. The DIF3D software could be used with the Intel and GNU compiler up to ~2013, but its use of some Fortran 66 features has eliminated compatibility with the latest GNU compilers and requires significant software changes to work with the latest GNU compilers.

The DIF3D software has no concept of data encapsulation and has heavy usage of common blocks for passing arrays and variables between subroutines. In almost every subroutine, the array bounds of every single or multi-dimensional array are not defined (i.e. uses "*"). This is despite the fact that array bounds are known from the variables passed in to the subroutine or available through common blocks. DIF3D uses implicit variable declarations in every subroutine and rarely are more than 20% of the variables used in a subroutine clearly defined as to their type and purpose in the subroutine. The character data is also stored in double precision arrays such that the end of string character is omitted. This requires DIF3D to constantly look for blank spaces to signify the end of a character string and, because it has a fixed format input option, requires the innumerable removal of spaces in read in names when trying to match named variables through the input processing and output generation.

BPOINTER is a memory addressing system built into DIF3D where multiple arrays are indexed into a single allocated contained obtained via a C malloc call. [61] BPOINTER was a revolution at the time as it allowed developers to use dynamic memory allocation to adjust the size of any execution to the necessary memory needed to run the calculation. Of course they immediately handicapped the BPOINTER implementation by requiring the user to specify, in their input files, the amount of memory that any job would take during execution instead of dynamically adjusting to the problem when running. Further, the original developers were very comfortable using BPOINTER as they consistently wrote subroutines that would use BPOINTER memory addressing evaluations at each consecutive layer of the source code instead of focusing the BPOINTER usage only at the highest level layers and passing in arrays and variables to define their sizes in a subroutine tree. In fact it is quite rare to have a subroutine in DIF3D that does not contain a common block of some form to allow additional usage of BPOINTER or access to control variables that developers did not bother to pass into the subroutine. Fortran Equivalence

statements are heavily used to define control variables from the arrays passed in via common block. While mostly done consistently across the entire code, there are numerous examples where the control variable naming changes from subroutine to subroutine.

In almost every solver of DIF3D, the memory computation is wrong in that the predicted amount of memory the code states it needs is not consistent with what is eventually used. This is a result of decades of updates by numerous software developers with no consistent adherence to the memory constraints imposed by BPOINTER. Of course, with a modern coding implementation, such calculations would not be needed as the software should just switch from one memory setup approach to another depending upon whether it fails to allocate the necessary memory to execute. Back fitting DIF3D to this type of model is not practical today given a constrained budget, but cleaning up the current implementation to a readable form that does not require the user memory inputs is manageable.

5.2.5 Compatibility

The DIF3D software has demonstrated usage on Linux, Windows, and macOS but only presently works with the Intel compiler. Significant source code changes were provided by Terrapower staff with regard to allowing DIF3D to work properly on Windows as there was no support funding from DOE. The BPOINTER component of DIF3D is a constant problem for source code operation, debugging, portability, and platform compatibility. Because of the reliability problems with the BPOINTER c malloc to Fortran translation on different compilers/platform a simple workaround was put in place to hardwire the BPOINTER container size. This of course is not compatible with Windows and thus future modifications are needed to overcome specific constraints placed on executables by the Windows operating system.

The BPOINTER infrastructure imposes that all of the arrays in DIF3D be mapped into a single container using 32 bit integers making BPOINTER a 2 gigabyte limited algorithm. While this works for most DIF3D reactor analysis projects, the abstraction of the user arrays inside of the DIF3D software make BPOINTER a constant problem for developers. Inside of BPOINTER, the software creates allows arrays of doubles, reals, 32 bit integers, and 8 character strings. To use any particular data type, the BPOINTER address mapping for each array has to be coded up uniquely in multiple locations throughout DIF3D as the array is not passed in common block, but by an abstract character string name. An additional problem with the BPOINTER methodology is the rare instance where usage overlaps between two adjacent arrays can lead to inaccurate answers in vector operations. In rare cases, a daughter subroutine is not aware that two passed in arrays are adjacent and when non-regular memory is accessed in a conventional vectorized scheme, it can inadvertently lock the storage for one of the two arrays near the overlap when both are intended to be modified. In these cases the algorithm fails to produce the correct value and random strange outputs result. Successful execution with disabled vectorization is always an indication of the problem and requires tedious debugging efforts to locate the subroutine and rewrite the subroutine to avoid the issue in the future.

The DIF3D code can perform sluggishly on large problems or when using large amounts of transport in DIF3D-VARIANT. These issues are traceable to the code design of the software to assume it has a very small amount of memory to work with. At numerous points in the code, it performs $O(N^3)$ searches to minimize memory usage where using a slightly larger amount of memory can reduce that search to order $O(N^2)$ if not O(N). As the problem size becomes large, the physical time required to sift through the large inputs becomes time consuming. Further, the software constrains branches of the code (such as the cross section homogenization) to work within very small, hardwired containers. To accomplish this, the developers use scratch binary files to store data during execution. As the problems sizes become large, these routines consume an enormous amount of time reading and writing data to disk that would trivially fit into memory.

The last problem is that the DIF3D software runs in serial. While threading is a very reliable way to improve the performance of most software packages, the early attempts to implement threading in DIF3D-VARIANT failed due to the way the developers constrained the algorithm to small memory chunks. In effect, the constant jumping from subroutine to subroutine to perform menial tasks make threading a net losing proposition for almost all problems attempted (less than 50% scaling). As the problem size increases, the serial DIF3D execution effectively makes what should be routine calculations into rarely attempted ones. While we have additionally researched acceleration techniques that can vastly improve the performance of DIF3D, implementing those schemes into DIF3D has always proved onerous because of the way the DIF3D code is setup and the lack of any funding support.

5.2.6 Critical Modeling Gaps

The fixable major problems with DIF3D that should be addressed are:

1) Elimination of the Fortran 66
2) Better documentation of the source code, its inputs and outputs
3) Elimination of artificial BPOINTER memory constraints
4) Correction of the peaking calculation
5) Inclusion of a flux error measure for fixed source problems

The Fortran 66 standard is 50+ years old. It is no surprise that compiler developers are refusing to support what amounts to spaghetti coding practices. In order to maintain the software in the following decade, it is going to become imperative that the code be cleaned up and at least made Fortran 77 standard. Given almost every subroutine in DIF3D has mixed Fortran 66 and Fortran 77, it will take months of work to carry out this process noting that scripts that automatically modernize older Fortran to Fortran 90+ have been attempted with unacceptable results.

The current manual is out of date. The creation of the manuals was always treated as an add-on operation as each consecutive solver was added to DIF3D. A single manual that discusses each solver as it is written today should be created. An additional document describing how to use the software and its coding structure should be created to better facilitate collaboration.

The BPOINTER infrastructure is a significant burden in the software to deal with but does not impact the actual design process other than annoying users. We can rather easily remove every BPOINTER array in the DIF3D code system by creating a single Fortran module that contains the BPOINTER arrays as actual Fortran 90+ allocatable arrays. This has already been demonstrated for DIF3D-VARIANT. In this scheme we would introduce the arrays as they actually are, globally accessible arrays stored in a (pseudo) common block. Identifying the parts of the code where the memory allocation needs are checked against the physical BPOINTER container size is actually more difficult than replacing the BPOINTER arrays. If we take this approach, we obviate the BPOINTER memory checking constraint by obviating using BPOINTER. (We actually set the arrays we are removing from BPOINTER to size 1 and then do not use them). Given that change, any user input would work as we would simultaneously eliminate the memory checks and any user input would work as there are only 500 or so arrays in the entire code system.

The larger problem to deal with is auto-adaptation of the DIF3D code to execute in different memory size mode. Much of the algorithm design in DIF3D is a result of the assumption of a low memory environment. To circumvent this problem and still allow the auto-adaptation requires us to unscramble and eliminate an enormous amount of Fortran 66 coding to allow the software to function properly after removing the arrays from BPOINTER. This requires a detailed knowledge of the constraints applied to the existing system instead of a single container constrained one. While this was successfully done for DIF3D-VARIANT in one aspect in 2009-2013, that work only focused on a few arrays and not the entire code system. The infrastructure in DIF3D that operates with BPOINTER is actually mixed in with the

timing routines and the file management structures, both of which are based upon deprecated Fortran 66 era programming schemes and would have to be updated as discussed in the previous issue.

The peaking calculation provided by DIF3D-FD is the only truly correct one produced by DIF3D. The one for DIF3D-Nodal is an approximation while the one for DIF3D-VARIANT is an unnecessary approximation. The intention of all three is to provide the user the peak fast flux or peak power within a given homogenized mesh. For fast spectrum systems, this will closely correspond to the peak pin power and peak fast flux. With a coarse mesh in DIF3D-FD, a typical approach taken by users, the typical peaking value can be an under prediction due to lack of sufficient mesh refinement. For DIF3D-Nodal, the approximation is derived from the lack of multi-dimensional spatial basis functions and thus is an approximation based upon the current on the interfaces. The same approximation used for DIF3D-Nodal is currently used in DIF3D-VARIANT although DIF3D-VARIANT has the necessary basis to exactly obtain the peak value. It is difficult to assess how inaccurate the DIF3D-Nodal estimates are, but a post analysis of the DIF3D-VARIANT results have shown that there is cause for concern.

The solution capability in DIF3D does not have an error check on the flux. Thus when one wants to solve a fixed source problem one has to guess how many iterations will be needed to solve the problem and then rerun the calculation against a higher or lower number of iterations to generate a comparison solution. Of course the comparison requires the user to look at the full 3D solution details to assess the errors. For licensing this is likely to be a significant problem as the checking procedure is a bit of a gray area. To resolve the issue, one must add about 1000 lines of code to DIF3D to resolve it. While this should have been added 30 years ago, it was not and still has not been corrected to this day.

5.2.7 User Support

The user support for DIF3D is unfunded and provided as needed when email contact is made with nera-software@anl.gov. There are several DIF3D users at Argonne who are qualified to provide support for DIF3D both in its use but also its compilation. The present user support is generally minimal and only provided by one staff member. What bug fixes do manage to get done are done at the expense of unassociated projects or done after normal work hours. At this point there is no plan to implement any of the requested user changes mentioned above and given the lack of support by DOE, there is little effort devoted to outreach or promotion of the DIF3D software.

A constant complaint identified by users are problems with the input data and the poor layout and documentation of the output. The current DIF3D input is based upon the original data card input structure where users would have 72 columns of input per card. To make the input as brief as possible, the developers loaded each line of input up with as many variables as possible. This means that instead of a logical, readable variable based input structure, the input is a series of numbers on lines that constantly require the user to refer back to the file format description to remember what each number means. Users make mistakes which cause the code to produce results that are rather easy to identify as wrong but that are annoying to have to continuously correct.

The input of physical data such as geometry and atom density are less of a burden with the control cards. For 3D hexagonal geometries, the DIF3D input approach is acceptable but for 3D Cartesian geometries it is terribly painful as changing the z directional information can require modifications to thousands of lines of input as the DIF3D input has users replace physical coordinates with integer indices. Modifying this structure is not doable, but it is practical to add a new input that allows a more convenient setup without the use of integer indices for the axial location of a given region. The major complaint with the remainder of the data input is that users are limited to six character names. It is a near trivial matter to upgrade the input processor to handle 8 character names which would alleviate the problem with complex user input significantly. However, it is not a trivial matter to upgrade all of the print statements, file formats, and connections in the code to handle 8 character strings instead of 6.

Most of the user complaints are focused on the output formatting. The existing formatting is associated with a historic practice of dumping the output directly to line printers instead of storing large data files. To accomplish that feat, all outputs from DIF3D are restricted to 80 columns of data. It is unheard of today for any user to directly print any aspect of a DIF3D input, instead preferring to copy/paste it into Excel or Word and print a much better formatted version. In this sense, it would be logical to modify the print out of DIF3D to eliminate the 80 column restriction for printing the flux, power, geometry and composition maps in addition to add convenient user desired features. These changes are actually rather easy to accomplish but requires detailed checking to ensure no errors are introduced and the elimination of enormous amounts of Fortran 66.

5.3 REBUS

The REBUS software is a fuel cycle analysis code that is wrapped around the DIF3D code. [74] The history of REBUS starts in the 1960s while the current version had the bulk of its code base assembled shortly after DIF3D was built in the late 1970s. Because REBUS is based upon DIF3D, it was designed as a homogenized region (not assembly) based fuel cycle code. REBUS requires the DIF3D input to be setup in a very specific way with regard to materials but no real changes with regard to geometry. In this manner, if a user provides a single region for the entire active core, REBUS will assume the user wants to homogeneously mix that region during the depletion process of each time step. If that is not the desired case, the user must introduce a sufficiently high enough number of spatial regions to eliminate the accuracy concerns associated with the homogenous mixing assumption. It is important to note that the homogeneous geometry assumption implicitly implies that the reaction rates are preserved on the average and not to any greater geometrical detail (flux reconstruction techniques are required).

REBUS has four built in depletion chains but most users always define their own depletion chain to provide REBUS the lumped fission product production rates and branching ratios that are appropriate for the spectrum of their reactor. The user defined depletion chain is a rather powerful feature of the REBUS code as it allows the maximum flexibility although extensive depletion chains can be very painful to manually setup.

The REBUS code has two primary operational modes: equilibrium and non-equilibrium mode. The non-equilibrium mode is the conventional approach taken for PWR analysis where the fuel is loaded into the core in specific positions, depleted for a specified time with intermediate shuffling, and discharged. Partially burned fuel assemblies can be stored in interim storage inside of REBUS as desired and used later in a fuel cycle operation. While the cross section treatments in REBUS are not as extensive as most PWR codes nor are thermal feedback changes over a cycle considered.

The REBUS equilibrium mode calculation is one of the unique features of REBUS compared with other codes. To quickly provide a fuel cycle analysis result for an idealized, repetitive fuel strategy, REBUS allows users to run this calculation with its equilibrium mode. In the equilibrium mode setup, the user must define a single burn step that considers the impact of shuffling fuel spatially in the domain. It does not calculate the flux solution at each unique loading, but smears together the fuel at each position after being subjected to the flux computed at each position. For fast spectrum systems, this approach rapidly yields a solution very similar to the shuffled fuel concept without the added computational expense. For PWR type thermal spectrum systems, this approach can yield a solution similar to the equilibrium behavior but only for homogeneous fuel concepts. The technique was developed for fast spectrum reactors and is not used for explicit assembly schemes, but groupings of assemblies such that detailed shuffling patterns do not have to be specified. In this manner, the users are able to simulate and study the impact of one area of the core having a 3 cycle fuel shuffling pattern while another area has a 4 cycle shuffling pattern without having to put together the detailed shuffling input. Such results always produce the idealized behavior of the reactor operating under the proposed loading scheme. The input is general such that the user can specify an arbitrary number of areas (termed fuel paths) and cycles per area (termed

batches). With respect to fast spectrum reactor design, it is one of the most important features of the code system to maintain.

REBUS was designed and built during the 1960s when breeder reactor technology was being heavily researched and thus fuel reprocessing was an important component of the success of that technology. Usable in both equilibrium and non-equilibrium modes, REBUS can be used to carry out fuel fabrication capabilities with general fertile and fissile feed options and general fuel recycling options. REBUS allows users to arbitrarily fabricate fuel for different regions of the core with regard to density and isotopic content. For reprocessing, REBUS can handle multiple recycling plants each operating with different efficiencies and time scales. With this capability, REBUS can be used to assess the feasibility and mass throughputs of an operating fleet of different reactors.

REBUS allows two fuel cycle search options all of which can be disabled as needed: enrichment or burnup limited cycle length. The enrichment search option is based upon a user specified separation of fuel isotopes in each feed assigned to each region with respect to the base (class 2) or enrichment material (class 1). The enrichment factor applied to adjacent regions can be further adjusted with region-wise multipliers. The general goal of the enrichment search is to achieve a targeted end of cycle criticality although one can target any chosen time point during the operating cycle. The burnup limit search will adjust the cycle length such that the discharge burnup from the specified path is met. Both of these searches can be disabled such that the REBUS code does a simple fuel cycle evaluation as desired by the user.

Even with the preceding brief description, it should be quite obvious that REBUS is a very complex piece of software. It has continuously been developed for 40 years and has 55+ years of actual research and development with regard to the source code and methodology. The fuel cycle modeling features that REBUS covers include the capabilities in the industry and some aspects that are unique to REBUS itself. The fact that there is no real replacement today and still is heavily used stands as a testament to its success and importance in fast reactor design work.

5.3.1 Software Quality Assurance

Much like DIF3D, there is no formal software control guidance document for REBUS. There is no written procedure for how the software is to be updated and distributed or how the reported software errors are to be addressed. Its quality and accuracy are a byproduct of the amount of free time available by the one active maintainer of the software. All software updates are applied by the current maintainer with no checking by any other staff member. There have been at least 20 contributors to the source code development in the last 40 years and the only constraint placed on any of those developers was to add new functionality as they like but do not break the existing functionality.

Given the age of the software, there are no component or algorithm tests of the REBUS software, only top-down tests. At present there are 28 test problems in the test suite for which reference outputs are provided. The same python based checking script used in DIF3D is used to detect unacceptable compiler/platform errors derived from compilation problems. Much like DIF3D, NEAMS funded the movement of the software into the same Subversion repository [51] in ~2008. In addition, the incorporation of multiple modifications from several developers were collected and merged into the main branch to create the current release.

The most recent manual was released in 1983 for version 3.0 of REBUS and thus REBUS is often called REBUS-3. [74] At issue is that there have been over 100 internal memos documenting changes to REBUS since 1983 but no official progression of the software version. This includes multiple releases of the software to RSICC with the same REBUS-3 name applied while the DIF3D version number progressed from version 3.0 to version 8.0. As part of the DOE-NEAMS work of 2008-2015, the various updates to the manuals, guidance documents on how to use the software and validation related documents were pulled together and released with the updated versions of DIF3D. The new features include input

changes to allow variable time steps, numerous bug fixes, modifications to allow modeling of accelerator driven systems and numerous additional outputs users can use to better assess fuel cycle performance. The version number applied to REBUS in the latest releases was 10.0 and later 11.0 to make the version number consistent with DIF3D as it is inherently an interconnected code system. (It was not possible to compile REBUS-3 against any version of DIF3D after 3.0 without making changes to REBUS). This choice was made solely by the only remaining active developer as it was simply convenient to release all of the software in a consistent manner.

The REBUS code solves the Bateman equations (depletion) with the flux solution from DIF3D on complex fuel cycle scenarios for a single plant. There are no analytical solutions of the Bateman equations past simplistic representations thus REBUS results must be compared against simple models with known physical behavior or against other codes which solve the same system. While this type of work was done in the past, it is poorly documented and not part of the current testing apparatus today. A considerable amount of work was done to produce validation cases for MC^2 + DIF3D + REBUS in the past, but those inputs have not been maintained and are poorly documented with regard to reproducibility. There are notable desired feature requests from users and several outstanding bugs that have not been addressed or resolved at this time.

5.3.2 Code Documentation

As stated previously, the current manual was written in 1983. [74] Its total length is 354 pages and only contains the discussion of a few output examples but all input options. The follow-on memos comprise almost 300 pages of text alone documenting new input features, new output features, resolution of code bugs and algorithm changes made at various points in the software development. To assemble the manual for the code as it exists today is a considerable undertaking noting that several sections within the code have been deprecated and there are physical inputs, described in the original document, that were created for the code which were never actually added to the code.

For the software methodology and usage, several power point guidance documents, originally part of the onsite training, are now provided to assist new users on how to properly use the software. While these are accurate, most users will still require an existing example input deck to build their own from or follow with regard to understanding how the software works.

Similar to DIF3D, there is no formal document that describes how to use the software, just a simple README file to understand how to execute for a given input in the distribution. There is no developers guide to provide assistance on the infrastructure design of REBUS. Beyond the few developers that already know how the software operates, extending the software beyond its current input and output usage is impractical for most potential developers or collaborators.

The input related errors caused by bad user inputs are generally well handled by REBUS although many users complain that the actual error with the input might have its source in a location other than where the fatal error output message implies. Some output produced by REBUS is poorly described and of dubious use with regard to a specific reactor analysis project and should be documented better.

5.3.3 Configuration Management

The REBUS software is export controlled and is housed in an export controlled Subversion repository. Its inclusion into the Subversion repository was a considerable modification of the source code to allow it to transition from a multi-step c-shell script driven maintenance system in SCCS on the deprecated Sun workstation system to a modern Linux based system with python based scripting. The Buildbot [73] service was setup to carry out nightly regression tests of the REBUS software since 2008 where more rigorous checking procedures were adopted in 2009-2010.

The approach to using Buildbot for REBUS is identical to that already described for DIF3D. The entire python testing apparatus is part of the software distribution and setup such that any person with the source code sees the repository exactly the way that it is maintained and execute the verification exactly the way the Buildbot software does it. While not perfect, this approach has minimized the potential for introducing software errors into the distribution.

5.3.4 Code Standards

The REBUS software contains a large amount of Fortran 66 with some Fortran 77 and no modern Fortran 90+. The REBUS software can only be used with the Intel compiler at this time due to an unidentified memory leak on Windows platforms and the fact that the GNU compilers do not support the Fortran 66 features in REBUS. Given this last aspect, each time a bug is fixed in REBUS, the practice has been to take the time to modify the subroutines to be Fortran 77 standard compliant. Even with this practice, we have only managed to update ~10% of the REBUS specific source code at this time.

The same issues that were pointed out with DIF3D exist in REBUS, but generally to a more severe degree. The DIF3D software was primarily written by three authors where the REBUS software has had at least 10. In this manner, one can distinguish different styles to documentation, memory management, and subroutine organization from the different authors. There are slight inconsistencies in the descriptions of some data arrays between authors although the usage is inevitably the same in order for the code to produce the correct result. Much like DIF3D, REBUS has no concept of data encapsulation and has heavy usage of common blocks for passing arrays and variables between subroutines. REBUS is the primary source of $O(N^3)$ search routines where simple $O(N^2)$ algorithms would suffice with the use of slightly more memory. These searches can cause the software to run considerably slow with large input decks with thousands of regions.

BPOINTER is heavily used in REBUS and an additional functionality was added where the BPOINTER contents were dumped to a binary file for restart. This avoids REBUS from having to reload input setup details by simply invoking a previous memory state that was valid. The follow-on RCT pin depletion code [75] relies upon this restart capability to execute its functionality and thus the output of this file must be maintained if validation of the software for depletion is to be performed. [76] While one could view this approach as a special binary file, the actual storage contents of the restart file are not necessarily consistent between different jobs. When the switch was made from Sun workstations to Linux-based computers, the restart capability of REBUS was permanently disabled as it did not function properly and there were no resources to correct it.

5.3.5 Compatibility

Since transitioning the REBUS software from Sun workstations to Linux-based computers, execution on Windows has not worked and the use with other compilers has generally failed. While one could blame this on bugs introduced during the transition process, it is more likely that the newer compilers and updated vectorization schemes along with the array overlap issue mentioned in the DIF3D section are to blame. The use of BPOINTER in REBUS is not exactly the same as DIF3D and thus debugging its use is considerably more difficult. The issues with input and output are worse in REBUS than in DIF3D and are a constant source of complaints from users.

5.3.6 Critical Modeling Gaps

The following critical gaps can be identified for REBUS:

1) Elimination of Fortran 66
2) Better documentation of the source code, its inputs, and outputs
3) Elimination of BPOINTER array usage

4) Correction of burnup output and better reporting of cycle peak flux and fluence details

As discussed for DIF3D, the Fortran 66 standard is outmoded and needs to be replaced. During this process, many of the input checking $O(N^3)$ loops will need to be replaced, although this is a much more involved process. While this particular issue does not prevent the software from being used, it unnecessarily wastes time and is relatively easy to fix. The goal of course is only to move the software from its current state to either Fortran 77 or higher standards.

The manual for REBUS is a considerable impediment to licensing. The current manual at best discusses half of what the software currently does and contains mistakes in several places. The fact that the user community use the software today in ways that are not described in the manual is more problematic as they will invariably become the dominant usage for licensing and thus are not documented. The numerous additional outputs that are commonly used today are also only documented in a series of informal internal memos which discuss the development changes more than the actual software functionality. The fact that some new changes are being made without consideration of any updates of the documentation pose additional problems.

The BPOINTER problems are a significant barrier in getting the REBUS software to be multi-compiler compatible at present. Much like the DIF3D case, the best path forward is to remove all of the existing arrays from the BPOINTER algorithm. This will require the creation of a new binary interface file for usage with RCT, but this is a minor issue compared with the changes required to obviating BPOINTER.

The fuel burnup, a common value in fuel cycle analysis, is reported is several places in the output. At least one of those places it is intended to be an approximate value based upon incomplete data. Many users mistakenly have taken this to be the burnup when the correct burnup results are provided on an auxiliary output file. This is poorly documented in the existing manual. Because the software was designed to print data directly to a line printer, it also produces intermediate results that occur during the convergence of the final pass through the fuel cycle in the same format as the final converged results (REBUS assumed low memory and thus repeats the flux calculation for each step instead of storing the previous result). This is very problematic as users constantly pick through the large, 60000 line outputs and pick the wrong section. While the use of scripts has greatly reduced these types of mistakes, it would be beneficial to produce a clear set of output files that collect the user data in a concise table form. The removal of the 80 column line output limit would also reduce the potential for errors by users. The final desired additional output is a better estimate of the peak burnup and fluence for each fuel path through the core which at present is just an approximation rather than a detailed follow which can be performed today given modern memory constraints.

5.3.7 User Support

The user support for REBUS is unfunded and provided as needed when email contact is made with nera-software@anl.gov. There are several REBUS users at Argonne who are qualified to provide support for using REBUS but only a smaller number on its compilation. The present user support is generally minimal and only provided by one staff member. What bug fixes do manage to get done are done at the expense of unassociated projects or done after normal work hours. At this point there is no plan to implement any of the requested user changes mentioned above and given the lack of support by DOE, there is little effort devoted to outreach or promotion of the REBUS software.

5.4 GAMSOR

GAMSOR is in essence a utility program to allow the calculation of gamma heating for use in the steady state thermal hydraulics code SUPERENERGY. [77] For fuel cycle analysis, the physical disposition of the gamma ray emissions from fission and non-fission neutron reactions is not relevant. Only that the total energy produced by fission is deposited somewhere in the domain. For thermal hydraulics, the gamma heating of structures poses a significant heat source throughout the domain that must be accounted. To

accomplish this, we must solve the coupled neutron-gamma transport equations. Note that we can neglect the gamma interactions that release neutrons and thus make this is a strictly one-way coupling.

The steady state neutron transport equation is commonly posed as an eigenvalue problem where the dominant eigenvalue gives an indication of the criticality of the system. The coupled gamma transport equation is only dependent upon the neutron reaction rates and not directly the eigenvalue itself. As a consequence, the coupled gamma transport equation can be solved as a fixed source problem after the neutron transport equation is solved. This is the GAMSOR sequence as it requires the user to carry out several steps in order to obtain the solution. It is easy to show that this is the most efficient approach that can be taken to solving the coupled system of equations.

The GAMSOR sequence involves three steps. The first step is to solve the reactor problem using a modified version of DIF3D. This step not only produces the neutron flux, but it also produces the fixed source file for the gamma flux problem. Given an accurate representation for the fixed source, the user must construct another DIF3D input and execute the fixed source solution of the gamma flux. There are notable problems with the fixed source solution capability that were discussed in the section on DIF3D. After these two steps are completed, the user must execute a summary DIF3D execution job such that two additional files will be created which are used in SUPERENERGY.

There is no real engineering output produced by GAMSOR other than the two binary files created during the third step. This is a major oversight in GAMSOR in that there is no readable power profile being produced by the software. Instead, the user must create their own utility to display the binary files (neutron and gamma power files) or rely upon SUPERENERGY to summarize the details for it.

5.4.1 Software Quality Assurance

There is no formal software control guidance document for GAMSOR. There is no written procedure for how the software is to be updated and distributed or how the reported software errors are to be addressed. Its quality and accuracy are a byproduct of the amount of free time available by the one active maintainer of the software. All software updates are applied by the current maintainer with no checking by any other staff member.

The GAMSOR software has six top down tests, four of which were created during the work on GAMSOR in 2016. None of these test problems were original tests for GAMSOR as none were created. Instead, they were just one past user's example problems that were used to construct input for SUPERENERGY. In this regard, the 2016 work put forth a considerable amount of effort to verify the accuracy of the procedure and do a code-to-code comparison of results against MCNP. While this is a continuous energy to multi-group test, the problem was made simple to ensure a rapid run time and accurate solutions in both codes. The details of that work are all discussed in the manual that was prepared for GAMSOR.

5.4.2 Code Documentation

A new manual was created for GAMSOR in 2016 where no real manual existed before. [77] It outlines the procedure to using the GAMSOR sequence and discusses the various aspects of the inputs and outputs. During the work to create the manual, the two power density files were added to the list of post processing files that the utility program DIF3D_TO_VTK could handle for visualization purposes. This new manual serves both as a description of the theory behind the GAMSOR solution process and the process of executing the software.

5.4.3 Configuration Management

The GAMSOR software is not export controlled but is labeled as applied technology. It is housed in an export controlled Subversion repository [51] as it is connected with other software that is exported controlled. Its inclusion into the Subversion repository was a considerable modification of the source code

to allow it to transition from a multi-step c-shell script driven maintenance system in SCCS on the deprecated Sun workstation system to a modern Linux based system with python based scripting. Unlike the rest of the source codes in this repository, GAMSOR was only added in 2013. The Buildbot [73] service was updated to carry out nightly regression tests of the GAMSOR software since 2013.

5.4.4 Code Standards

The GAMSOR source code was constructed in the late 1980s and inexplicably contains Fortran 66, Fortran 77 and one subroutine written in Fortran 90 during the work in 2016. The only real Fortran 66 inclusions are Hollerith format statements which were deprecated with the Fortran 77 standard but the developers chose to include them in their work anyway. Because GAMSOR is more of a series of utility programs, a bulk of its functionality is buried into the SUMMAR module of DIF3D itself and only invoked with specific input files. As a consequence, most of the coding involves special calls to deal with the modified input and thus fixing DIF3D would invariably resolve any issues with GAMSOR.

5.4.5 Compatibility

GAMSOR, like DIF3D, has demonstrated usage on Linux, Windows, and macOS but only presently works with the Intel compiler. Given its similarities and heavy use of DIF3D subroutines, it also has considerable use of BPOINTER and thus the same problems as DIF3D does. Since GAMSOR runs DIF3D for the steady state problem, its execution is dominated by the time consumed by DIF3D to solve for the neutron flux.

5.4.6 Critical Modeling Gaps

Because of the work in 2016, the only real fixable issue with GAMSOR is the removal of the remaining Fortran 66 and resolution of the BPOINTER related issues. These issues do not constitute any more of a burden than those already described for DIF3D. One could consider building a newer utility function to replace GAMSOR and provide users with a more convenient to use capability with actual engineering outputs. This updated version would orchestrate the existing piecemeal operations performed by the users and still produce the desired coupled data for SUPERENERGY.

5.4.7 User Support

The user support for GAMSOR is unfunded and provided as needed when email contact is made with nera-software@anl.gov. There are only a few GAMSOR users at Argonne who are qualified to provide support for GAMSOR both in its use and compilation. The present user support is generally minimal and only provided by one staff member. Although there has only been one bug identified during the 2016 work, any future bugs would be done at the expense of unassociated projects or done after normal work hours. Because of the work done in 2016, there is no plan to implement any of the requested user changes mentioned above, and given the lack of support by DOE, there is little effort devoted to outreach or promotion of the GAMSOR software.

5.5 SUPERENERGY

SUPERENERGY is often referred to in publications as SUPERENERGY-2-ANL or SE2-ANL. [78] The SUPERENERGY-2 software was the development outcome of work done creating the code sequence of ENERGY, ENERGY-II, ENERGY-III, and SUPERENERGY in the mid 1970s. With each consecutive software development, more accuracy and capability was added to the software such that the final version, SUPERENERGY-2, produced acceptable results for sodium cooled fast spectrum reactors with hexagonal pitched assemblies. [43] Originally produced by PNNL and MIT, the delivered software was connected to the Argonne suite of reactor codes in the early 1980s and heavy modification was applied all the way through 2008. (The latest changes were made to consider oxide fuel instead of just metal fuel like

that done previously). In addition to coupling the software to DIF3D via GAMSOR, the ANL developers added a fuel pin model, computed hot channel factor associated cladding temperature predictions, and included an automatic orificing scheme. Because none of the original source codes are distributed today and all that remains is SUPERENERGY-2-ANL, we just refer to it as SUPERENERGY or SUPERENERGY 11.0 such that its version is consistent with the DIF3D release to which it is coupled.

The primary advantage of SUPERENERGY is that it is able to quickly compute the steady state thermal-hydraulic details of each fuel assembly bundle in a full reactor core model. The geometry assumption is an array of wire-wrapped fuel pins from the neutronics problem inlet to the neutronics problem outlet. There is no consideration of inlet plenum or outlet plenum geometry. Each fuel assembly is assumed to consist of a wire-wrapped pin lattice that has no gaps due to thermal expansion. SUPERENERGY will generally run in less than 2 minutes for really large reactor core problems and has been shown to be very accurate with regard to experimental measurements.

5.5.1 Software Quality Assurance

There is no formal software control guidance document for SUPERENERGY. There is no written procedure for how the software is to be updated and distributed or how the reported software errors are to be addressed. Its quality and accuracy are a byproduct of the amount of free time available by the one active maintainer of the software. All software updates are applied by the current maintainer with no checking by any other staff member.

Source code control on SUPERENERGY appears to have been lost at least twice during the period of use of SUPERENERGY at ANL. The initial loss occurred in the transfer from PNNL/MIT to ANL where there are only hand written notes describing the original software provided to ANL and the modifications made to connect it with the ARC software. The second instance was in the early 2000s where no tracking of any source code modifications were made to obtain the current version. While we do have a previous version in the SCCS repository, to determine what changes were made to the source code and their impact would require a considerable comparison between the new and old source codes. The primary issue with losing source code control is that any validation work done previous to the source code control cannot be believed on the current version and thus either has to be redone or the impact of the interim source code changes must be addressed. There is no current reason to believe that the SUPERENERGY accuracy was diminished as a result of the interim source code changes.

The version of SUPERENERGY maintained on the Sun workstation (before the 2002 source code control loss) had a single verification test problem derived from the PRISM reactor design work. This development corresponds to the work done on SUPERENERGY to include BPOINTER into SUPERENERGY. The test problems before that version were created were likely those constructed by the original authors for CRBR. The current SUPERENERGY software has seven top down tests created during the design work for ABR and the Korean fast spectrum PGSFR. There are rarely good reasons for switching the verification test problems and thus the quality of the solutions from SUPERENERGY require some level of assessment to ensure they are accurate. It is important to note that this type of assessment has to be carried out in any reactor design work, regardless the status of the software, as the correlations built into SUPERENERGY all have to be verified as valid for the reactor type of interest.

5.5.2 Code Documentation

The software documentation is quite poor for SUPERENERGY. While the original developers did provide a manual describing the software implementation, the theory section of the manual is particularly unclear and has handwritten equations. A rather low-level literature survey on the history of SUPERENERGY turned up 25 published journal papers and technical laboratory reports of relevance. In those documents, the actual implementation details of SUPERENERGY show it transformed from a porous body medium methodology to a sub-channel methodology that is in SUPERENERGY today. Such

a transformation had a dramatic impact upon the code input requirements and thus we cannot entirely rely upon previous publications to determine the set of equations and the methodology of how they are solved in SUPERENERGY today. The code also suffers from inputs that do not work and solve options that do not work. Experienced users have learned how to avoid the known problems with running their calculations. In summary, the SUPERENERGY software is poorly documented by any standard.

5.5.3 Configuration Management

The SUPERENERGY software is not export controlled but is labeled as applied technology. It is housed in the same export controlled Subversion repository [51] that DIF3D is housed in for convenience as it is connected to DIF3D via GAMSOR. Its inclusion into the Subversion repository required rather minor source code changes. Unlike the rest of the source codes in this repository, SUPERENERGY was only added in 2013. The BuildBot [xxx] service was updated to carry out nightly regression tests since 2013.

5.5.4 Code Standards

The SUPERENERGY source code was written in the mid 1970s and thus is Fortran 66. The later modifications were written in Fortran that is sometimes more primitive than Fortran 66 and in many cases identically Fortran 66 where the explanation given was "to keep with the existing style" whereas using Fortran 77 would have saved a lot of extra effort. Because we have access to the SUPERENERGY source code before the 1990s work to add BPOINTER, it was clear that the original version had compile-time hardwired arrays. While this structure may have seemed inconvenient at the time, adding BPOINTER actually made the code harder to read as it introduces abstracted arrays mapped into a single container. Ignoring these aspects, the four or five source code contributors did create a piece of software that is rather consistent throughout, with sufficient comments to understand the purpose of each subroutine.

5.5.5 Compatibility

SUPERENERGY has demonstrated usage on Linux. It has a lot of Fortran 66 constructs and uses BPOINTER and thus has the same compiler related problems as the software described earlier.

5.5.6 Critical Modeling Gaps

Any thermal-hydraulic code that does not involve CFD has correlations embedded into it. SUPERENERGY is not a CFD code and thus its accuracy is bound by the accuracy of the geometry assumptions that go into it and the correlations that are used. While we can complain about its usage of BPOINTER, unlike the other software discussed thus far, the use of BPOINTER in SUPERENERGY is relatively benign and can be replaced with a few days of work in a manner similar to that discussed in other sections of this report. Given this, the following critical gaps can be identified for SUPERENERGY:

1) Elimination of Fortran 66
2) Better documentation of the source code, its inputs, and outputs

The presence of Fortran 66 is a significant barrier for compatibility of the source code. It is relatively easy to modify SUPERENERGY to eliminate the Fortran 66 in favor of Fortran 77 as it is mostly focused on the use of Hollerith statements more than other deprecated features.

The code documentation must be done in concert with defining a proper theory manual. At present it is not clear what exact system of equations are being solved and how all of the different correlations are being pulled in. For any type of licensing activity, there will likely be new experiments carried out that correspond to the new fuel assembly design. It would rather useful to be able to incorporate such new correlations into the existing code. If such experiments are not practical, then it might be wise to just incorporate correlations from later time periods instead of just those available in the early 1970s. In

addition to these changes, the output of the software could be better presented and basic visualization capabilities added.

5.5.7 User Support

The user support for SUPERENERGY is unfunded and provided as needed when email contact is made with nera-software@anl.gov. There are two users of SUPERENERGY at Argonne who are qualified to provide support for using it. However, because ANL does not export the software, the present user support is generally only provided from one user to the other. What bug fixes do manage to get done are done at the expense of unassociated projects or done after normal work hours. At this point there is no plan to implement any of the requested user changes mentioned above and given the lack of support by DOE, there is little effort devoted to outreach or promotion of the SUPERENERGY software.

5.6 PERSENT

PERSENT is a perturbation and sensitivity code built upon the DIF3D-VARIANT solver. [79] Its predecessor was VARI3D which was built upon the DIF3D-FD solver. [46] VARI3D was built around the general need to produce point kinetics reactivity coefficients needed in safety analysis codes and validate those reactivity coefficient measurements against physical experiments. A sensitivity calculation capability was later added to VARI3D for the DIF3D-FD RZ geometry option for assessing cross section uncertainties. The creation of PERSENT came as a result of known problems with the accuracy of diffusion theory based reactivity coefficients. PERSENT is thus primarily focused on generating the point kinetics based reactivity coefficients needed in safety analysis and providing cross section sensitivity coefficients for several key parameters needed in uncertainty quantification. PERSENT can be considered a complete replacement of VARI3D except for some of the perturbation and sensitivity options in VARI3D that are of key interest to integral measurements taken in experiments. Given that the U.S. has no operating fast reactor to produce experimental data, the development of the specialized functionalities was deemed unnecessary at the time of development.

PERSENT not only can provide equivalent diffusion theory reactivity coefficients to VARI3D, but also can produce transport theory based reactivity coefficients. The current implementation uses the compiled DIF3D executable as an externally called program from PERSENT making the PERSENT software separate from DIF3D. While there is some linkage to subroutines in DIF3D, PERSENT is generally an independent piece of software. The bulk of the computational effort in PERSENT is actually the time required for DIF3D to provide the steady state forward or adjoint flux solutions.

5.6.1 Software Quality Assurance

There is no formal software control guidance document for PERSENT. There is no written procedure for how the software is to be updated and distributed or how the reported software errors are to be addressed. Its quality and accuracy are a byproduct of the amount of free time available by the one active developer of the software. All software updates are applied by the current maintainer with no checking by any other staff member.

PERSENT is written in Fortran 90+ where a bulk of the actual coding would be considered Fortran 95. PERSENT relies heavily on DIF3D for its software quality assurance as its primary purpose is to carry out matrix-vector operations based upon the solutions provided by DIF3D-VARIANT. There are unit tests for the module components that PERSENT includes, but those unit tests do not contain complex functions and can be summarized as checking allocation/deallocation of memory or reading/writing files that DIF3D produces. During its initial creation, 16 test problems were included, but a more recent revision to improve the performance of the sensitivity calculations includes 18 verification tests. For perturbation theory problems, the inputs were taken from the DIF3D benchmark test suite and are easy to validate assuming DIF3D-VARIANT can produce an accurate solution to any given input problem.

Several test problems check the first order perturbation theory option, a key component for generating reactivity coefficients. The primary means of checking is to perform a direct perturbation and compare the result to the first order perturbation result. As the direct perturbation decreases in magnitude, it will identically produce the first order perturbation result noting that accuracy diminishes as the perturbation reaches the iterative convergence limits.

Similar to perturbation theory, the sensitivity verification problems include a set of analytical problems and a set of larger reactor sized problems for verification. The reference solutions for the larger problems were obtained using the finite difference methodology for sensitivity calculations which was documented in the manual as are the analytical problems. The manual also discusses the larger scale ABR problem using 21 groups which was verified to be accurate using the finite difference methodology.

In summary, the PERSENT code is relatively easy to ensure accuracy as it relies upon DIF3D to be accurate. The only complex result to check is the first order perturbation which is done with the existing set of verification problems by performing the same perturbation using progressively smaller perturbations. In that regard, the linear derivatives in PERSENT are rather easy to validate using any set of test problems for both perturbation and sensitivity theory.

5.6.2 Code Documentation

The PERSENT manual is actually a manual for VARI3D and PERSENT where VARI3D is distributed with PERSENT. The manual details the theory for PERSENT for both perturbation and sensitivity theories. The same methodology is applied in VARI3D, but the additional perturbation and sensitivity functionalities are not discussed in the PERSENT manual. The manual clearly discusses the input and output for PERSENT along with most of the verification problems and the results that they produce.

5.6.3 Configuration Management

The PERSENT code is considered to be export controlled but no formal review has been carried out and no formal classification applied. It is housed in an export controlled Subversion repository [51] but it is not connected with any other software that is exported controlled. Its inclusion into the Subversion repository was natural as the repository existed prior to its construction and thus the source code development is part of the commit logs in the repository. The Buildbot [73] service was updated to carry out nightly regression tests of the PERSENT software since its creation in ~2013. VARI3D was an original component of the transition and had to undergo significant debugging to overcome the problems with compiler compatibility and its obtuse use of Fortran 66. It has been under Buildbot regression since ~2009.

5.6.4 Code Standards

PERSENT is written in Fortran 90 named files where a bulk of the actual coding could be considered Fortran 95. Because PERSENT is interlinked with DIF3D, the data structuring associated with DIF3D binary files (Fortran 90 modules) is used heavily to pass multiple arrays between subroutines. Nevertheless, the total number of arrays needed by PERSENT is large and a common block approach Fortran module was constructed which contains the PERSENT specific arrays. While this common block should have been avoided, the lack of real connection in the array allocation made it difficult to discern which arrays should be collocated in a data structure and thus a unique module. In time, this common block approach will likely be removed although it does not really amount to much of an issue as it is only used to avoid passing all of the arrays through large subroutine headers from the driver level of the code to the first few levels of the PERSENT code.

5.6.5 Compatibility

PERSENT can compile and run on Linux or macOS and there should be no problems with Windows compilation. As for compilers, PERSENT has been tested with GNU but the inability to compile DIF3D with the GNU compiler precludes its ability to be used with that compiler at this time.

5.6.6 Critical Modeling Gaps

There are no critical gaps at this point with regard to PERSENT's functionality or use that can be identified at this time. The software is rather easy to manipulate to get whatever desired functionality is needed such as adding in the other perturbation and sensitivity options that VARI3D contains although we cannot conceive of a need to do that anytime soon. This assessment is primarily based upon the fact that what PERSENT does is simple algebraic manipulations of the results that DIF3D produces.

5.6.7 User Support

The user support for PERSENT is unfunded and provided as needed when email contact is made with nera-software@anl.gov. There are several users of PERSENT at Argonne who are qualified to provide support for using it. However, because ANL does not export the software, the present user support is generally only provided from the one developer. What bug fixes do manage to get done are done at the expense of unassociated projects or done after normal work hours. At this point there is no plan to implement any new features given the lack of support by DOE and there is little effort devoted to outreach or promotion of the PERSENT software.

5.7 SAS4A/SASSYS-1

SAS4A/SASSYS-1 (hereafter, SAS) is a software simulation tool used to perform deterministic analysis of anticipated events as well as design basis and beyond design basis accidents for advanced fast reactors. [25] Detailed, mechanistic models of steady-state and transient thermal, hydraulic, kinetic, and mechanical phenomena are employed to describe the response of the reactor core, the reactor primary and secondary coolant loops, the reactor control and protection systems, and the balance-of-plant to accidents caused by changes in coolant flow, loss of heat rejection, or reactivity insertion.

The consequences of single and multiple-fault accidents can be modeled, including fuel and coolant heating, fuel and cladding mechanical behavior, core reactivity feedbacks, coolant loop performance including natural circulation, and decay heat removal. Analyses are typically terminated upon demonstration of reactor and plant shutdown to permanently coolable conditions, or upon violation of design basis margins. The objective of the analysis is to quantify accident consequences as measured by the transient behavior of system performance parameters, such as reactivity feedback, fuel and cladding temperatures, margins to coolant boiling, and cladding strain.

Originally developed for analysis of sodium cooled reactors with oxide fuel clad by stainless steel, the models in SAS are being extended and specialized to metallic fuel clad with advanced alloys and to several other coolant options, including lead and lead-bismuth eutectic.

Perhaps the strongest factor that influenced early fast reactor safety analysis was the concern over the possibility of core compaction followed by an energetic core disassembly — the so-called Bethe-Tait accident. [80] In the late 1960s, the Hanford Engineering Development Laboratory (HEDL) began developing the MELT code [81] [33] to evaluate the initiating phase of hypothetical core disruption accidents (HCDA) as part of the FFTF project. The MELT series of codes had the capability to model the transient behavior of several representative fuel pins (channels) within a reactor core to allow for incoherency in the accident sequence. By 1978 MELT had evolved into the MELT-IIIB code. [33]

Around the same time that development on MELT began, Argonne National Laboratory began developing the SAS series of codes. [82] [83] [84] [85] [86] [87] Like MELT, SAS has the capability to

model the transient behavior of several representative channels to evaluate the initiating phase of HCDAs. SAS1A [82] originated from a sodium boiling model and includes single- and two-phase coolant flow dynamics, fuel and cladding thermal expansion and deformation, molten fuel dynamics, and a point kinetics model with reactivity feedback. By 1974, SAS evolved to the SAS2A computer code [83] which included a detailed multiple slug and bubble coolant boiling model which greatly enhanced the ability to simulate the initiating phases of loss-of-flow (LOF) and transient overpower (TOP) accidents up to the point of cladding failure and fuel and cladding melting.

The SAS3A code [84] added mechanistic models of fuel and cladding melting and relocation. This version of the code was used extensively for analysis of accidents in the licensing of FFTF. In anticipation of LOF and TOP analysis requirements for licensing of the Clinch River Breeder Reactor Plant (CRBRP), new fuel element deformation, disruption, and material relocation models were written for the SAS4A version of the code, [85] which saw extensive validation against TREAT test data. In addition, a variant of SAS4A, named SASSYS-1, [86] was developed with the capability to model ex-reactor coolant systems to permit the analysis of accident sequences involving or initiated by loss of heat removal or other coolant system events. This allows the simulation of whole-plant dynamics feedback for both shutdown and off-normal conditions, which have been validated against EBR-II Shutdown Heat Removal Test (SHRT) data and data from the FFTF LOF tests.

Although SAS4A and SASSYS-1 were originally maintained as two computer codes, they had always shared a common code architecture, the same data management strategy, and the same core channel representation. Subsequently, the two code branches were merged into a single code referred to as SAS4A/SASSYS-1. From that point, “dotted” version numbers were adopted, but the previous version identifiers (e.g. “4A”) remain embedded in the name. Version 2.1 of the SAS4A/SASSYS-1 code was distributed to Germany, France, and Japan in the late 1980s, and it serves as a common framework for international oxide fuel model developments.

Beyond the release of Version 2.1, revisions to SAS continued throughout the Integral Fast Reactor (IFR) program between 1984 and 1994, culminating with the completion of SAS v 3.0 in 1994. [87] In the latter part of this time period, the modeling emphasis shifted towards metallic fuel and accident prevention by means of inherent safety mechanisms. The whole-plant dynamics capability of the SASSYS-1 component plays a vital role in predicting passive safety feedback. Without it, meaningful boundary conditions for the core channel models are not available, and accident progression is not reliably predicted.

By the mid 1990s, SAS v 3.1 had been completed as a significant maintenance update, but because of the termination of the IFR program, it was not released until 2012. [88] In the time since the original development of Version 3, several modeling additions and enhancements have been made to meet U.S. DOE programmatic needs. Important among these are

- Detailed sub-channel models for whole-core analyses to resolve intra-assembly temperature and flow distributions, [89] along with 3D visualization capabilities for sub-channel results.
- Extended decay-heat models to support long-term transients and complex, actinide-bearing fuels. [90]
- Support for coupling with external CFD simulations to resolve flow distribution and thermal stratification effects. [91] [92] [93]
- Support for spatial kinetics (requires DIF3D-K).

The above updates, although significant, were sporadic. Nevertheless, they served the critical role of maintaining some level of development and ensured the code could be passed to the next code manager.

Around 2010, industry began expressing interest in SAS for supporting advanced fast reactor development needs. In response, Version 5.0 was released in 2012. [88] The release incorporated all of the previously unreleased updates. Since then, a major restructuring of the code has been completed to

adapt all source files to free-form source format and new model developments are being implemented using modern object-oriented practices. [16] [17] [18] [19] [20]

5.7.1 Software Quality Assurance

One response to the Safety Licensing Research Plan was the creation of the Regulatory Technology Development Plan. [4] An objective of the RTDP was to assess SFR computer code capabilities and the activities required to support qualification for regulatory use. Over the course of the program (see Figure 2), [7] a software quality assurance program was formally established for SAS and went into effect in March 2018. [8]

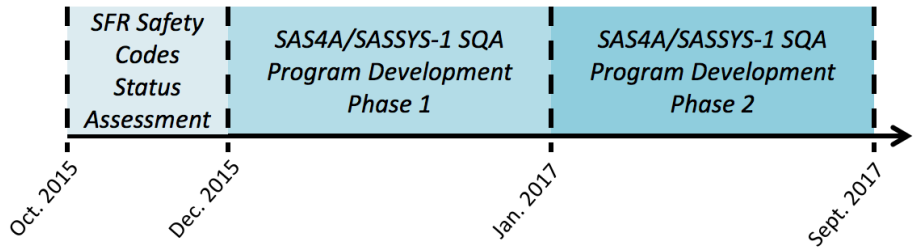


Figure 2: Development Timeline for the SAS Software Quality Assurance Program

The SQA Program for SAS consists of the elements shown in Figure 3. The SQA Plan (SQAP) comprises the top level document and provides an overview of all program activities and requirements. Additional elements of the program include the following:

- **Configuration Management Plan** (CMP): This is a second-tier document that describes configuration management, which is the process of identifying, managing, and controlling the status and revision of software items. Configuration management activities are inherent in all procedures.
- **Coding Standards**: This is a second-tier document that contains requirements for programming practices and conventions. It identifies relevant standards that are inherent to all SQA activities.
- **Procedures**: These are third-tier documents. The following procedures are defined for the SAS SQA Program:
  - **Software Testing** defines steps for evaluating whether software adequately performs all intended functions.
  - **Error Reporting and Corrective Actions** defines steps for identifying, reporting, and correcting errors.
  - **Software Development and Modification** defines steps for the design and implementation of new features or models.
  - **Version Release** defines steps for the formal release of software and delivery of products.

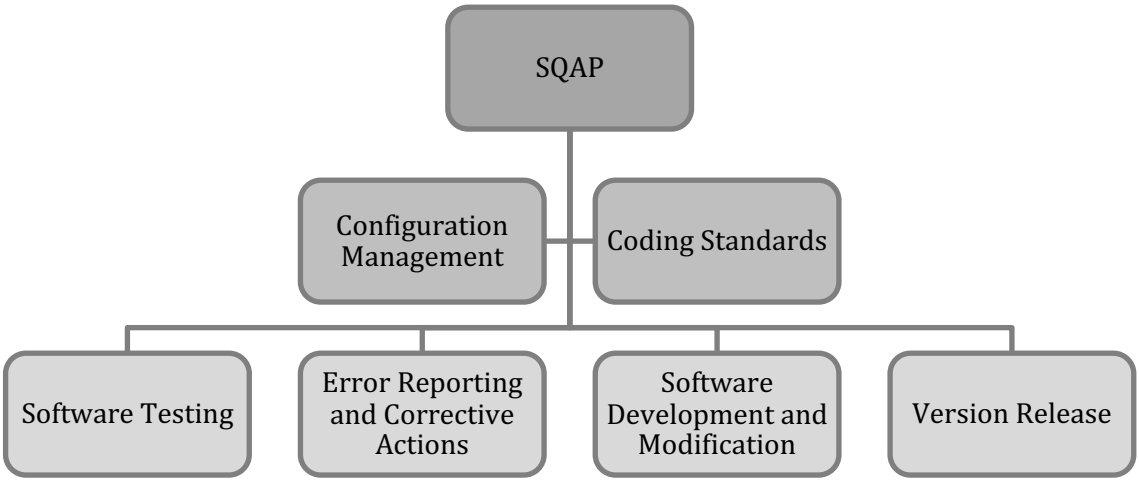


Figure 3: The SAS Software Quality Assurance Program Hierarchy

5.7.2 Code Documentation

The most recent manual for SAS was released in 2017, [25] yet it is based almost entirely on the documentation completed in 1996. [87] In published form, the manual consists of approximately 2000 pages and includes detailed descriptions of the equations, discretization, and solution methods used. Due to the complexity of the modeling features in SAS for fast reactor accident analysis, considerable expertise is required to understand the documentation. For expert users and developers, the manual is a valuable resource.

Unfortunately, the SAS manual is more of a code manual than a user manual. It lacks instructional material such as tutorials and examples. In addition, the sheer size of the manual has become a major barrier to maintenance in its current form. The process of removing outdated information or adding updated content is difficult and not amenable to SQA procedures.

A new task has been initiated to convert the manual into a plain text markup format (reStructuredText [94]) that can be used to generate web-based content with more powerful searching and cross-referencing capabilities. Text-based content is far more convenient for configuration management, modification (inserting, deleting, merging content), and review. Although considerable progress has been made, issues related to converting proprietary equation formats to open-source LaTeX [95] markup have hindered completion.

5.7.3 Configuration Management

Configuration management for SAS is formally implemented under the SQA Program. Items maintained under configuration control include source code, developer utilities, test cases and reference outputs, user utilities for post-processing, and documentation. All of these items are maintained in a Subversion repository [51] that requires authenticated access. Changes to configuration items are tracked along with the author, date, time, location, and description of each change.

SQA activities are documented in the ticket system within a Trac environment. [96] Activities include the development of new code features, error reporting and corrective actions, modifications to documentation, changes to the build environment, and new version releases. The ticket system has been configured to support the work flow shown in Figure 4, thus providing documentation of the approval process for each ticket.

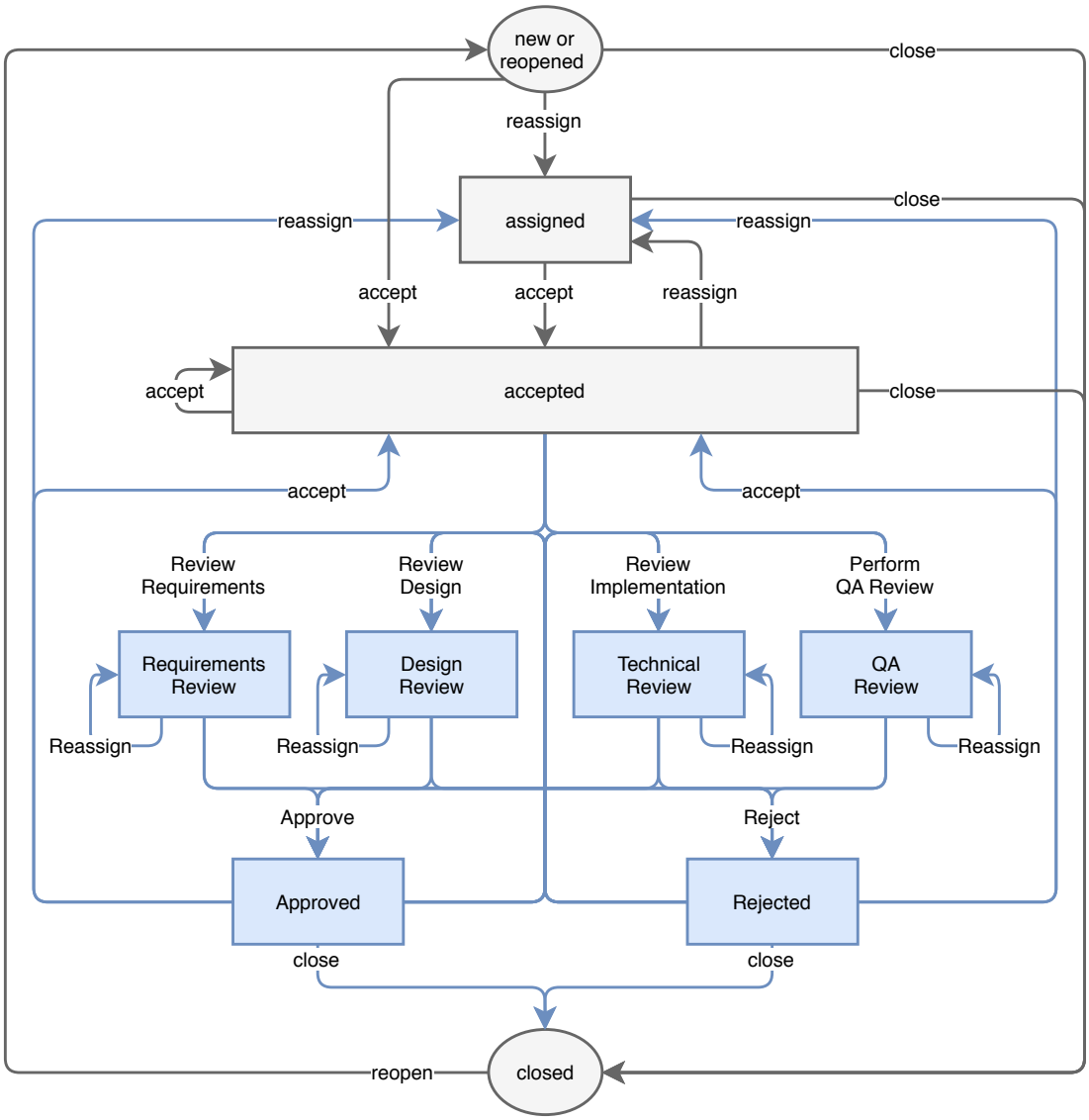


Figure 4: Ticket Workflow Defined for the SAS Trac Environment.

5.7.4 Code Standards

Coding standards have been established for SAS with the goal of establishing consistency, readability, and portability to improve code quality and to support overall life-cycle maintenance. Although the requirements do not retroactively apply to portions of the code that existed prior to the implementation of the SQA Plan, the majority of the SAS source code is now compliant with Fortran 2003, C99, and C++03 language standards. [97] [98] [99]

Compliance with modern language standards does not mean that the source code is in a form that promotes readability and maintenance. While deprecated language elements have been eliminated, most of the source has minimal commenting and formatting. Lack of *if-then* blocks in the Fortran 66 language standards meant that previous developers were more comfortable using complex *goto* logic combined with a morass of labeled statements, and the current status of the code reflects that style.

The coding standards for SAS include additional requirements for style, variable and interface declarations, numerics, and comments. It is important to continue maintenance and development activities so that these requirements can be implemented in larger portions of the code.

5.7.5 Compatibility

The code standards described above help ensure cross-platform compatibility. Currently, SAS can be executed on Intel-based x86 platforms running Linux, macOS, or Windows. The major limitation in terms of compatibility is with compilers. The Intel Fortran compiler is currently required for generating executables and non-Intel platforms are not supported.

Recent attempts at compiling with the GNU Fortran compiler [100] result in a compiler crash. This indicates a problem with the compiler, not necessarily with the SAS source code. There has been very limited success with using the NAG Fortran compiler, [101] however simulation results do not exactly match reference solutions. This may be due to different optimization strategies used by the different compilers, or it could be due to flaws in code implementation. Because it is time consuming to maintain development activities with multiple compilers and operating systems, the root cause of these differences has not been identified.

5.7.6 Critical Modeling Gaps

Improvements needed in SAS include not only the implementation of new capabilities required to support safety analyses, but source code changes required to improve usability, performance, scalability, and maintenance. The latter is an ongoing process that is required to maintain existing capabilities and developer expertise as programming languages and computational resources evolve. The former is based on the prioritization of identified gaps in modeling capabilities. Critical gaps, based on current R&D activities, include the following:

Metal Fuel Transient Modeling: The Korea Atomic Energy Research Institute (KAERI) has invested considerable resources to develop the metallic fuel accident modeling capabilities in SAS. [102] [103] [104] [105] [106] Initial development of these models began around 1990 but never completed, yet they continue to be directly relevant to U.S. SFR designs. Currently, however, these models exist in a separate branch of code development. Considerable effort is needed to optimize the performance of these models and to reintegrate the updates into the official SAS code.

Oxide Fuel Transient Modeling: International organizations, such as CEA (Commissariat à l'énergie atomique et aux énergies alternatives) and JAEA (Japan Atomic Energy Agency), have interest in oxide fuel severe accident modeling. SAS already has a strong oxide fuel modeling capability, but other organizations have made improvements to the models. JAEA is working with Argonne to determine which improvements are essential for incorporation into SAS. This work needs to continue so that DOE maintains its leadership role in international fast reactor safety.

Support for UQ and Dynamic PRA Analysis: Support for using SAS to perform transient uncertainty quantification (UA) and sensitivity analysis has recently been developed. [107] [108] [109] [110] Similarly, SAS was used to support the modernization of the PRISM probabilistic risk assessment (PRA). [111] [112] [113] [114] [115] [116] [117] Even though SAS is a low-fidelity code, these types of applications can quickly generate terabytes of data. The methods used in these studies have not been integrated into SAS. Related to this work, SNL developed preliminary code modifications to support dynamic PRA analyses with SAS. [118] Unfortunately, there are insufficient resources to continue this work. UQ and PRA capabilities are expected to be more important for future authorization or licensing activities that employ a risk-based approach, and work should continue in these areas.

5.7.7 User Support

Within Argonne, developers and analysts maintain very close collaborations. Users outside of Argonne are generally expected to establish a funded mechanism to receive support and training. These mechanisms have included Strategic Partnership Programs (SPP), Cooperative Research and Development Agreements (CRADA), Gateway for Accelerated Innovation in Nuclear (GAIN) vouchers, Nuclear Energy University Partnership (NEUP) awards, and Funding Opportunity Announcement (FOA) awards.

Along with the success in deploying SAS to universities, industry, and international collaborators, there are criticisms. Universities have significant challenges in establishing expertise and in funding training, and domestic industry has expressed interest in participating in user groups. Both of these issues could be addressed if a user group meeting could be organized on a regular basis. Instead, developers and expert users at Argonne attempt to address questions by e-mail as time permits.

Finally, despite the existence of a formal SQA Program, a documented mechanism for reporting issues has not yet been established. Users need to be able to report issues to developers, and developers need to alert affected users of potential problems. A means for facilitating this communication should be established along with support for user group interactions.

6 Major Gaps

Based on the preceding assessment of each code against the defined goals, major gaps can be identified. Gaps are ranked in priority — high (critical), medium, or low — according to their potential to be an impediment for an authorization basis or license application process. This ranking is summarized graphically in Figure 5.

It is unsurprising that lack of an SQA program for six of the codes is a major gap. Due to lack of maintenance resources during the same time that formal SQA standards evolved, formal programs have not been established or maintained for MC^2, DIF3D/VARIANT, REBUS, GAMSOR, SUPERENERGY, and PERSENT. Due to support from the RTDP Program, a formal SQA program was established for SAS in 2018.

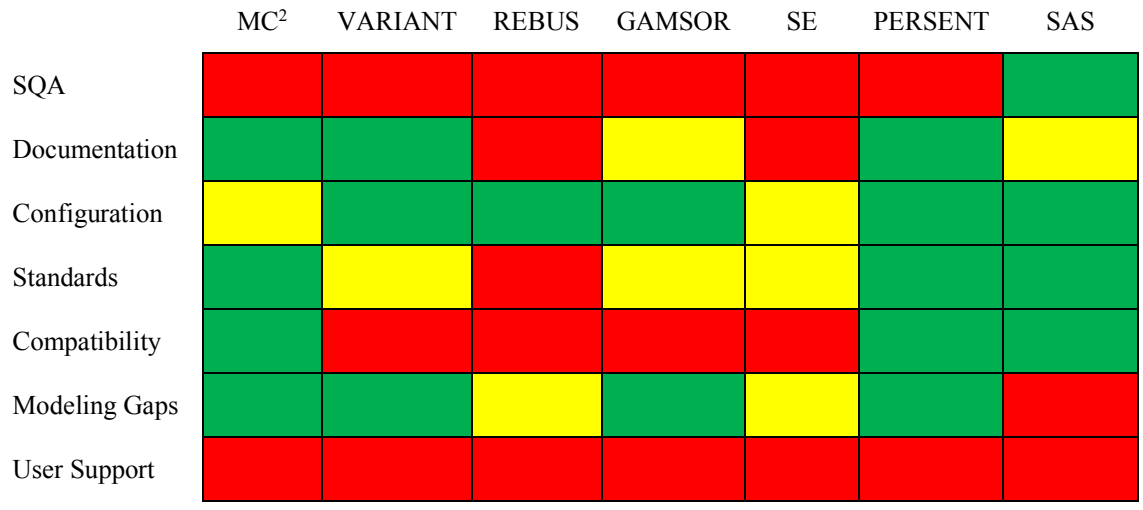


Figure 5: Categorization of Gaps for Each Code as High (red), Medium (yellow), or Low (green).

Two codes stand out in terms of having medium or high priority gaps under most, if not all, of the goals: REBUS and SUPERENERGY. Both lack sufficient documentation for the implementation and use of the fuel cycle and thermal analysis capabilities, respectively. Functionality of the codes is preserved mainly by institutional knowledge shared between users. REBUS is used far more often than SUPERENERGY, and therefore has a stronger knowledge base. Due to insufficient maintenance resources, both have significant gaps in terms of language standards and cross-platform compatibility, and can therefore only run on one or two platforms.

Compatibility is an issue for all of the DIF3D-based codes — VARIANT, REBUS, GAMSOR, and SUPERENERGY — due primarily to the outdated nature of memory management. As described earlier in this report (e.g. see Section 5.2.5), all of these codes were originally developed for small memory machines. Pointer manipulation and data caching schemes significantly impair maintainability, scalability, and portability.

SAS has a notable modeling gap. Initial code development for metal fuel transient modeling began around 1990, but it was never completed. More recently, KAERI has invested considerable resources to continue this development, however the updates exist in a separate branch of the code. Additional resources are needed to improve the performance of these models and to reintegrate the updates into the official SAS distribution.

A user-support environment does not exist for any of the codes and support is, at best, *ad-hoc*. Developers and users of the codes do not have access to issue reporting and resolution mechanisms, training opportunities, or collaboration resources.

Finally, Figure 5 illustrate the benefit of investments in MC^2, PERSENT, and SAS. MC^2 and PERSENT have received some funding from the NEAMS Program. MC^2 Version 3 is a significant update from Version 2, and PERSENT is a new code to replace VARI3D. However NEAMS no longer supports these codes. SAS has received funding from the ART Program to modernize the code structure and from the RTDP Program to implement an SQA Program.

7 Recommendations

Based on the assessment of the seven codes documented in Section 5 against the goals defined in Section 4, several priorities emerge. At a fundamental level, the state of development and maintenance is abysmal. Four recommendations are made below that, if followed, will significantly improve the state of these codes and ensure their continued availability to support DOE programmatic objectives.

7.1 Establish Development Team

For all of the codes required for fast reactor design and safety analysis, there are essentially only four developers. Of those, only two have experience with modern software engineering practices. With the exception of SAS, the remaining codes have little or no funding to support maintenance activities. Instead, issues are triaged at the expense of other programmatic needs. Maintenance of the codes relies on random, buckshot-style funding from numerous sources, including laboratory directed research and development (LDRD) projects, International Nuclear Energy Research Initiatives (INERI), bilateral collaborations, and strategic partnership programs (SPP) with domestic and international organizations. Functionality of the current codes is due in great part to the commitment and motivation of three code managers who have dedicated considerable free time to ensure continued viability.

In the past, development of these codes relied on dozens of staff and several code managers. Past teams comprised a major portion of a single Division. Fortunately, the maturity of the software methodology, advances in computing capabilities, and availability of modern code management systems and automatic regression testing apparatus means such large teams are not required to sustain the software. But a team does need to exist.

It is recommended that DOE establish coherent work packages for the purpose of developing and maintaining fast reactor design and safety analysis methods. A work package for SAS modernization already exists, and it could be expanded to support all of the codes identified in this report. The total level of effort required for proper maintenance is estimated at five full-time equivalent staff, supporting a team of eight to ten developers, far lower than pre-1994 levels. The team should reflect a combination of expertise in numerical methods, reactor physics, reactor safety, and software engineering.

Without a team of sufficient size, it is not possible to preserve knowledge of the implemented codes, address critical modeling gaps, perform technical reviews, respond to reported issues, or support code qualification for an authorization basis or license application.

7.2 Implement Software Quality Assurance

SAS is the only fast reactor analysis code currently maintained under a software quality assurance program. The *development and implementation* of the SAS SQAP was funded by the RTDP program, but that funding ended in 2017. *Maintenance* of the SQAP now falls on the developers. Continuing an SQA Program requires support for Program manager responsibilities such as interfacing with Laboratory and DOE QA programs, managing SQA training requirements, performing reviews, and supporting management assessments and potential audits. In addition, SQA programs require a well-defined workflow for code development activities with associated documentation, approvals, and technical reviews.

It is recommended that DOE place a priority on establishing SQA programs for the remaining fast reactor design and analysis methods. It should be relatively straightforward to establish an SQA Program for the remaining codes. Based on the experience in developing the SQA Program for SAS, the effort required is estimated to be equivalent to one full-time staff over two years. After an SQA Program is implemented, maintenance would then fall on the development team. Unlike SAS, the other codes are under a different export control classification and reside on a different server. That server doesn't allow reconfiguration of the workflow to match SQA requirements, so moving the codes and related configuration management databases to a different server may be required.

7.3 Create User Support Environment

Support for the *use* of fast reactor analysis methods should be funded by the programs that rely on the codes. This is the current practice. However, as described in Section 4.7, support for the user-base also needs to include formal issue reporting and response mechanisms, training opportunities, collaboration resources, and user-group meetings. Domestic and international users have expressed a strong interest in learning from shared expertise.

It is recommended that DOE establish a fast reactor methods user group that encourages collaboration between users with different levels of expertise. This type of environment would support knowledge preservation among geographically dispersed users and provide direct user feedback to developers. As part of these activities, code manuals should be extended to include more user-centric content such as examples and tutorials. Funding requirements for this should be modest as users would fund their own participation. However, additional funds could be provided to support participation by university students and professors. This would help establish a pipeline of expertise that will benefit both DOE and domestic industry.

7.4 Close Critical Modeling Gaps

Assuming that a Work Package has been established to support a fast reactor methods development team and that appropriate Software Quality Assurance practices are in place, critical gaps need to be addressed. Numerous gaps have been identified in Section 5. Based on the potential impact these gaps have on code

sustainability or on an authorization basis or license application, three have been selected as having the highest priority.

It is recommended that DOE place priority on closing the following critical gaps:

- Implement missing transient phenomena for metal fuel. Transient modeling with SAS feeds directly into accident analyses for an authorization basis or licensing application. Missing phenomena include metallic fuel restructuring and growth, sodium bond logging and relocation, transient fuel property characterization, in-pin fuel melting and relocation, cladding failure, and fuel and fission gas ejection mechanics. Code to address these phenomena has been developed under the PGSFR program sponsored by KAERI, but considerable effort is needed to improve computational performance and to integrate models into the production version of SAS.
- Eliminate BPOINTER from all DIF3D-based codes. The current memory management scheme violates language standards and inhibits portability, scalability, and compatibility with computing platforms and compilers. With the availability of multi-core, large memory computing resources, the past limitations that led to the development of BPOINTER no longer apply. Instead, its presence poses a threat to the sustainability of the highly optimized numerical methods implemented in DIF3D as computing platforms continue to evolve.
- Improve compatibility of REBUS on more platforms. REBUS provides unique fuel cycle analysis capabilities not available in any other code, yet it suffers the most from lack of maintenance. The code does not function on Windows and documentation is extremely limited. Like all DIF3D-based codes, it relies on BPOINTER. However, the underlying implementation differs and different approaches may be needed to eliminate the dependency.

The preceding list summarizes the gaps judged to be the most critical, but numerous others have been documented that should be prioritized and addressed in the future.

8 Summary

The development of fast reactor design and safety analysis methods spans nearly six decades. During that time, significant progress has been made in refining the numerical methods used to represent the physics occurring during steady-state and transient conditions. Perhaps the most active period of development was from the late 1960s to the early 1990s. As a result of U.S. policy changes in 1994, development came to an abrupt halt.

Despite the termination of formal development support, maintenance activities continued in an ad-hoc fashion in order to preserve the extensive capabilities that existed. Lack of updates, combined with a rapid evolution in computing technologies, led to a fragile code system supported by an aging development team. After a renewed interest in fast reactor technologies, new code managers were assigned and source code was placed under more modern configuration management to prevent complete loss. However, resources available for code maintenance were still extremely sparse.

In 2011, Sandia National Laboratories published the Safety Licensing Research Plan that evaluated the state of code capabilities for supporting a fast reactor license application. The panel of experts "expressed concern that US code development activities had stagnated and that the experienced user-base […] was decaying away quickly." Subsequently, the U.S. DOE established a work package to begin modernizing the SAS4A/SASSYS-1 safety analysis code. Additional resources from the NEAMS program supported significant improvements to the fast spectrum cross-section processing code MC^2 and the development of a new code, PERSENT, for performing transport-theory perturbation and sensitivity analyses.

In 2015, Idaho National Laboratory published the Regulatory Technology Development Plan that identified additional gaps that needed to be addressed to support regulatory acceptance of DOE-sponsored research, whether it be software or data. Subsequently, the U.S. DOE established work packages to assess

capabilities to perform source term calculations, to establish quality control over metallic fuel performance data, and to establish a software quality assurance program for SAS4A/SASSYS-1.

This document serves as a Fast Reactor Methods Development Plan. It extends the previous two studies by focusing specifically on the software simulation tools that support the fast reactor design and safety analysis activities required for an authorization basis or license application. Explicit goals have been defined relating to maintenance, development, and use of these tools. Seven critical computer codes used in the analysis work flow have been evaluated against these goals to identify specific gaps.

Although development activities are no longer stagnant, they are insufficient to support an authorization basis or license application. In order to preserve the knowledge and capabilities inherent in these codes, an adequately funded development team needs to exist. The top priorities of the development team should be to establish formal software quality assurance practices for all of the codes and to resolve critical gaps that could be impediments to design, authorization, or license applications. It is also important to recognize that development and maintenance is a separate activity from the application of the codes by reactor designers and safety analysts. Where the two intersect is in user support. An environment that fosters interaction between developers and all users should be established.

9 Acknowledgements

Argonne National Laboratory's work was supported by the U.S. Department of Energy, Office of Nuclear Energy, under contract DE-AC02-06CH11357.

10 References

[1] M. Denman, J. LaChance, T. Sofu, G. Flanagan, R. Wigeland and R. Bari, *Sodium Fast Reactor Safety and Licensing Research Plan — Volume I*, SAND2012-4260, Sandia National Laboratories, May 2012.

[2] R. Schmidt, T. Sofu, T. Wei, J. Thomas, R. Wigeland, J. Carbajo, H. Lidewig, M. Corradini, H. Jeong, F. Serre, H. Ohshima and Y. Tobita, *Sodium Fast Reactor Gaps Analysis of Computer Codes and Models for Accident Analysis and Reactor Safety*, SAND2011-4145, Sandia National Laboratories, June 2011.

[3] D. Grabaskas, M. Bucknor and T. Sofu, *Assessment of Regulatory Technology Gaps for Advanced Small Modular Sodium Fast Reactors*, ANL-SMR-9, Argonne National Laboratory, 2014.

[4] Idaho National Laboratory, *Advanced Reactor Technology — Regulatory Technology Development Plan (RTDP)*, INL/EXT-14-32837, Idaho National Laboratory, 2015.

[5] A. J. Brunett and L. L. Briggs, *unpublished information*, Argonne National Laboratory, 2015.

[6] N. R. Brown, W. D. Pointer, M. T. Sieger, G. F. Flanagan, W. Moe and M. Holbrook, *Qualification of Simulation Software for Safety Assessment of Sodium-Cooled Fast Reactors: Requirements and Recommendations*, ORNL/TM-2016/80, Oak Ridge National Laboratory, April 2016.

[7] A. J. Brunett, L. L. Briggs and T. H. Fanning, *Status of SFR Codes and Methods QA Implementation*, ANL-ART-83, Argonne National Laboratory, January 2017.

[8] A. J. Brunett and T. H. Fanning, *Implementation of Software QA for SAS4A/SASSYS-1*, ANL-ART-110 Rev. 1, Argonne National Laboratory, March 2018.

[9] Y. S. Tang, R. D. Coffield Jr. and R. A. Markley, Thermal Analysis of Liquid Metal Fast Reactor Reactors, Westinghouse Electric Corporation, Madison, Pennsylvania: American Nuclear Society Publications, 1978.

[10] U.S. Department of Energy, *A Compendium of Computer Codes for the Safety Analysis of Fast Breeder Reactors*, DOE/ET-0009, U.S. Department of Energy, 1977.

[11] A. Yacout, *personal communications*, Argonne National Laboratory, 2018.

[12] D. Grabaskas, A. Brunett, M. Bucknor, J. Sienicki and T. Sofu, *Regulatory Technology Development Plan Sodium Fast Reactor — Mechanistic Source Term Development*, ANL-ART-3, Argonne National Laboratory, February 2015.

[13] D. Grabaskas, M. Bucknor and J. Jerden, *Regulatory Technology Development Plan Sodium Fast Reactor — Mechanistic Source Term – Metal Fuel Radionuclide Release*, ANL-ART-38, Argonne National Laboratory, February 2016.

[14] D. Grabaskas, M. Bucknor, J. Jerden and A. Brunett, *Regulatory Technology Development Plan Sodium Fast Reactor — Mechanistic Source Term – Trial Calculation*, ANL-ART-49, Argonne National Laboratory, October 2016.

[15] T. H. Fanning, F. E. Dunn, D. Grabaskas, T. Sumner and J. W. Thomas, *SAS4A/SASSYS-1 Modernization*, ANL-ARC-265, Argonne National Laboratory, September 2013.

[16] T. H. Fanning, A. J. Brunett, T. Sumner and N. Stauff, *SAS4A/SASSYS-1 Modernization and Extension*, ANL-ARC-299, Argonne National Laboratory, September 2014.

[17] T. H. Fanning, A. J. Brunett, T. Sumner and R. Hu, *SAS4A/SASSYS-1 Modernization and Extension*, ANL-ART-30, Argonne National Laboratory, September 2015.

[18] T. H. Fanning, A. J. Brunett and G. Zhang, *SAS4A/SASSYS-1 Code Improvements for FY2016*, ANL-ART-75, Argonne National Laboratory, September 2016.

[19] T. H. Fanning and A. J. Brunett, *Status of the SAS4A/SASSYS-1 Safety Analysis Code*, ANL-ART-97, Argonne National Laboratory, June 2017.

[20] T. H. Fanning, *FY2017 Updates to the SAS4A/SASSYS-1 Safety Analysis Code*, ANL-ART-122, Argonne National Laboratory, September 2017.

[21] L. L. Humphries and D. L. Y. Louie, *MELCOR/CONTAIN LMR Implementation Report – FY14 Progress*, SAND-2014-19183, Sandia National Laboratories, October 2014.

[22] L. L. Humphries and D. L. Y. Louie, *MELCOR/CONTAIN LMR Implementation Report – FY15 Progress*, SAND-2016-0484, Sandia National Laboratories, January 2016.

[23] L. L. Humphries and D. L. Y. Louie, *MELCOR/CONTAIN LMR Implementation Report – FY16 Progress*, SAND-2016-12101, Sandia National Laboratories, November 2016.

[24] D. L. Y. Louie and L. L. Humphries, *Development of a MELCOR Sodium Chemistry (NAC) Package – FY17 Progress*, SAND-2018-1252, Sandia National Laboratories, February 2018.

[25] T. H. Fanning, A. J. Brunett, T. Sumner and (eds.), *The SAS4A/SASSYS-1 Safety Analysis Code System*, ANL/NE-16/19, Argonne National Laboratory, Nuclear Engineering Division, March 2017.

[26] A. M. Yacout and M. C. Billone, "Current Status of the LIFE Fast Reactors Fuel Performance Codes,", Palais des Congrès, Paris, France, March 4–7," in *Int'l Conf. on Fast Reactors and Related Fuel Cycles (FR13)*, Paris, France, March 2013.

[27] A. Boltax and et al., in *Int'l Fast Reactor Safety Meeting*, Snowbird, 1990.

[28] U.S. Department of Energy, *Software Quality Assurance Improvement Plan: MACCS2 Gap Analysis*, DOE-EH-4.2.1.3-MACCS2-Gap Analysis, U. S. Department of Energy, Office of Environment, Safety, and Health, May 2004.

[29] C. J. Werner and (ed), *MCNP Users Manual - Code Version 6.2*, LA-UR-17-29981, Los Alamos National Laboratory, 2017.

[30] Sandia National Laboratory, *MELCOR Computer Code Manuals, Vol. 1: Primer and Users' Guide, Version 2.2.9541*, SAND 2017-0455 O, Sandia National Laboratory, January 2017.

[31] M. T. Farmer, J. J. Sienicki, B. W. Spencer and C. C. Chu, "Status of the MELTSPREAD-1 Computer Code for the Analysis of Transient Spreading of Core Debris Melts," in *2nd CSNI Specialist Meeting on Core Debris-Concrete Interactions*, Karlsruhe, Germany, April 1–3, 1992.

[32] A. E. Walter and et al., *MELT-III: A Neutronics, Thermal-Hydraulics Computer Program for Fast Reactor Safety Analysis*, HEDL-TME-74-47, Handford Engineering Development Laboratory, December 1974.

[33] K. K. Tabb, C. H. Lewis, L. D. O'Dell, J. A. Padilla, D. E. Smith and N. P. Wilburn, *MELT-IIIB: An Updated Version of the MELT Code*, unpublished information, Hanford Engineering Development Laboratory, April 1979.

[34] S. S. Tsai, *The NACOM Code for Analysis of Postulated Sodium Spray Fires in LMFRBs*, NUREG/CR-1405, Brookhaven National Laboratory, 1980.

[35] G. A. McLennan, *NUBOW-3D (Inelastic): A FORTRAN Program for the Static Three-Dimensional Structural Analysis of Bowed Reactor Cores Including the Effects of Irradiation Creep and Swelling*, ANL-CT-78-19, Argonne National Laboratory, 1978.

[36] J. Grudzinski and T. Moran, *Supplement to the NUBOW-3D Manual*, ANL/NE-15/9 Rev. 1, Argonne National Laboratory, 2015.

[37] A. G. Croff, *A User's Manual for the ORIGEN2 Computer Code*, ORNL/TM-7175, Oak Ridge National Laboratory, July 1980.

[38] S. Ludwig, *A Revision to ORIGEN2 - Version 2.2*, transmittal memo, Oak Ridge National Laboratory, May 2002.

[39] U.S. Nuclear Regulatory Commission, *SCDAP/RELAP5/MOD 3.3 Code Manual*, NUREG/CR-6150, U.S. Nuclear Regulatory Commission, August 2000.

[40] U.S. Nuclear Regulatory Commission, *TRACE V5.0 Users Guide*, ADAMS Accession No. ML120060403, U.S. Nuclear Regulatory Commission, June 2008.

[41] W. R. Bohl and L. B. Luck, *SIMMER-II: A Computer Program for LMFBR Disrupted Core Analysis*, LA-11415-MS, Los Alamos National Laboratory, 1990.

[42] P. Beiriger, J. Hopenfeld, M. Silberberg, R. P. Johnson, L. Baurmash and R. L. Koontz, *SOFIRE II User Report*, AI-AEC-13055, Atomics International, March 1973.

[43] K. L. Basehore and N. E. Todreas, *SUPERENERGY-2: A Multiassembly, Steady-State Computer Code for LMFBR Core Thermal-Hydraulic Analysis*, PNL-3379, Pacific Northwest Laboratory, August 1980.

[44] Y. W. Shin, C. K. Youngdahl, H. C. Lin, B. J. Hsieh and C. A. Kot, *User's Manual for the Sodium-Water Reactor Analysis Computer Code SWAAM-II*, ANL-83-75, Argonne National Laboratory, 1983.

[45] C. H. Adams, *Specifications for VARI3D—A Multidimensional Reactor Design Sensitivity Code*, FRA-TM-74, Argonne National Laboratory, 1975.

[46] M. A. Smith, C. Adams, W. S. Yang and E. E. Lewis, *VARI3D & PERSENT: Perturbation and Sensitivity Analysis*, ANL/NE-13/8, Argonne National Laboratory, June 2013.

[47] J. Jackson and R. Nicholson, *VENUS-II: An LMFBR Disassembly Program*, ANL-7951, Argonne National Laboratory, September 1972.

[48] *IEEE Standard for System, Software, and Hardware Verification and Validation*, IEEE Std 1012-2016, IEEE, 2016.

[49] M. J. Rochkind, "The Source Code Control System," in *IEEE Trans. on Software Engineering*, 1975.

[50] W. Tichy, "Design, Implementation, and Evaluation of a Revision Control System," in *ICSE 1982 Proc. of the 6th Int'l Conf. on Software Engineering*, 1982.

[51] B. Fitzpatrick, C. Pilato and B. Collins-Sussman, Version Control with Subversion, O'Reilly Media, June 2009.

[52] *X3.9-1966: Fortran*, X3.9-1966, American National Standards Institute, New York, New York, 1966.

[53] *ISO/IEC 1539-1:2010: Programming Languages — Fortran — Part 1: Base Language*, ISO/IEC 1539:2010, 2010.

[54] *C 2011: Programming Languages — C*, ISO/IEC 9899:2011, 2011.

[55] *C++2017: Programming Languages — C++*, ISO/IEC 14882:2017(E), 2017.

[56] C. H. Lee and W. S. Yang, "Development of Multigroup Cross Section Generation Code MC2-3 for Fast Reactor Analysis," in *Proc. of Int. Conf. on Fast Reactors and Related Fuel Cycles (FR09)*, Kyoto, Japan, December 2009.

[57] C. H. Lee and W. S. Yang, *MC2-3: Multigroup Cross Section Generation Code for Fast Reactor Analysis*, ANL/NE-11-41 Rev. 2, Argonne National Laboratory, January 2012.

[58] C. H. Lee and W. S. Yang, "Verification and Validation of Multigroup Cross Section Code MC2-3 for Fast Reactor Systems," in *Trans. American Nuclear Society*, 2012.

[59] C. H. Lee and W. S. Yang, "An Improved Resonance Self-shielding Method for Heterogeneous Fast Reactor Assembly and Core Calculation," in *Proc. Int'l Conf. Mathematics and Computational Methods Applied to Nuclear Science and Engineering (M&C 2013)*, Sun Valley, ID, May 2013.

[60] H. Henryson II, B. J. Toppel and C. G. Stenberg, *MC2-2: A Code to Calculate Fast Neutron Spectra and Multigroup Cross Sections*, ANL-8144, Argonne National Laboratory, 1976.

[61] L. C. Just, H. Henryson II, A. S. Kennedy, S. D. Sparck, B. J. Toppel and P. M. Walker, *The System Aspects and Interface Data Sets of the Argonne Reactor Computation (ARC) System*, ANL-7711, Argonne National Laboratory, 1971.

[62] K. L. Derstine, *DIF3D: A Code to Solve One-, Two-, and Three-Dimensional Finite-Difference Diffusion Theory Problems*, ANL-82-64, Argonne National Laboratory, 1984.

[63] R. D. McKnight, *Benchmark Testing Using ENDF/B VERSIONS III and IV**, ZPR-TM-214, Argonne National Laboratory, 1975.

[64] R. D. Lawrence, *The DIF3D Nodal Neutronics Option for Two- and Three-Dimensional Diffusion Theory Calculations in Hexagonal Geometry*, ANL-83-1, Argonne National Laboratory, March 1983.

[65] R. D. Lawrence, "Progress in Nodal Methods for the Solution of the Neutron Diffusion and Transport Equations," in *Prog. Nuclear Energy*, 1986.

[66] G. Palmiotti, E. E. Lewis and C. B. Carrico, *VARIANT: VARIational Anisotropic Nodal Transport for Multidimensional Cartesian and Hexagonal Geometry Calculation*, ANL-95/40, Argonne National Laboratory, 1995.

[67] C. B. Carrico, E. E. Lewis and G. Palmiotti, "Three Dimensional Variational Nodal Transport Methods for Cartesian, Triangular and Hexagonal Criticality Calculations,," *Nuclear Science and Engineering,* vol. 111, p. 168, 1992.

[68] G. Palmiotti, C. B. Carrico and E. Lewis, "Variational Nodal Transport Methods with Anisotropic Scattering," *Nuclear Science and Engineering,* vol. 115, p. 233, 1993.

[69] E. E. Lewis, C. Carrico and G. Palmiotti, "Variational Nodal Formulation for the Spherical Harmonics Equations," *Nuclear Science and Engineering,* vol. 122, p. 194, 1996.

[70] E. E. Lewis, C. B. Carrico and G. Palmiotti, "Variational Nodal Formulation for the Spherical Harmonics Equations," *Nuclear Science and Engineering,* vol. 122, p. 194, 1996.

[71] M. A. Smith, N. Tsoulfanidis, E. E. Lewis, G. Palmiotti and T. A. Taiwo, "Higher Order Angular Capabilities of the VARIANT Code," in *Trans. American Nuclear Society*, 2002.

[72] M. A. Smith, E. E. Lewis and E. R. Shemon, *DIF3D-VARIANT 11.0, A Decade of Updates*, ANL/NE-14/1, Argonne National Laboratory, 2014.

[73] B. Warner, "Buildbot: The Continuous Integration Framework," [Online]. Available: http://docs.buildbot.net/1.2.0/full.html. [Accessed 28 June 2018].

[74] B. J. Toppel, *A User's Guide to the REBUS-3 Fuel Cycle Analysis Capability*, ANL-83-2, Argonne National Laboratory, 1983.

[75] W. S. Yang and M. A. Smith, *RCT: REBUS Based Pin Power Reconstruction Using the DIF3D-Nodal and DIF3D-VARIANT Options*, ANL/NE-14/15, Argonne National Laboratory, December 2014.

[76] R. D. McKnight, *Validation of the Rebus-3/RCT Methodologies for EBR-II Core-Follow Analysis*, ANL-IFR-153, Argonne National Laboratory, September 1991.

[77] M. A. Smith, C. H. Lee and R. N. Hill, *GAMSOR: Gamma Source Preparation and DIF3D Flux Solution*, ANL/NE-16/50 revision 1.0, Argonne National Laboratory, June 2017.

[78] J. C. Beitel, *Status of and Revised Input for SE2-ANL*, unpublished information, Argonne National Laboratory, January 1985.

[79] M. A. Smith, W. S. Yang, A. Mohamed and E. E. Lewis, "Perturbation and Sensitivity Tool Based on the VARIANT Option of DIF3D," in *Trans. American Nuclear Society*, San Diego, CA, November 2012.

[80] H. A. Bethe and J. H. Tait, *An Estimate of the Order of Magnitude of the Explosion When the Core of a Fast Reactor Collapses*, RHM-56-113, April 1956.

[81] A. E. Waltar, A. P. Jr. and R. J. Shields, *MELT-II, A Two-Dimensional Neutronics-Heat Transfer Program for Fast Reactor Safety Analysis*, unpublished information, Hanford Engineering Development Laboratory, 1972.

[82] J. C. Carter and et al., *SAS1A, A Computer Code for the Analysis of Fast-Reactor Power and Flow Transients*, ANL-7607, Argonne National Laboratory, October 1970.

[83] F. E. Dunn and et al., *The SAS2A LMFBR Accident-Analysis Computer Code*, ANL-8138, Argonne National Laboratory, October 1974.

[84] M. G. Stevenson, W. R. Bohl, F. E. Dunn, T. J. Heames, G. Hoppner and L. L. Smith, "Current Status and Experimental Basis of the SAS LMFBR Accident Analysis Code System," in *Proc. Fast Reactor Safety Meeting*, Beverly Hills, CA, April 1974.

[85] G. Birgersson and et al., *The SAS4A LMFBR Accident Analysis Code System*, ANL/RAS 83-38, Argonne National Laboratory, February 1988.

[86] F. E. Dunn and et al., *The SASSYS-1 LMFBR Systems Analysis Code*, ANL/RAS 84-14, Argonne National Laboratory, March 1987.

[87] Reactor Analysis Division, *The SAS4A/SASSYS-1 LMR Analysis Code System*, ANL-FRA-1996-3, Argonne National Laboratory, August 1996.

[88] T. H. Fanning and (ed), *The SAS4A/SASSYS-1 Safety Analysis Code System*, ANL/NE-12/4, Nuclear Engineering Division, Argonne National Laboratory, January 2012.

[89] F. E. Dunn, J. E. Cahalan, D. Hahn and H. Y. Jeong, "Whole Core Sub-Channel Analysis Verification with EBR-II SHRT-17 Test," in *Proc. 2006 Int'l Congress on Advances in Nuclear Power Plants*, Reno, NV, June 2006.

[90] T. H. Fanning, J. W. Thomas, T. Sumner and A. J. Brunett, "Recent Developments for the SAS4A/SASSYS-1 Safety Analysis Code," in *Proc. 16th Int'l Topical Mtg. on Nuclear Reactor Thermal Hydraulics (NURETH-16)*, Chicago, IL, August 2015.

[91] T. H. Fanning and T. Sofu, "Modeling of Thermal Stratification in Sodium Fast Reactor Outlet Plenums During Loss of Flow Transients," in *Proc. International Conference of Fast Reactors and Related Fuel Cycles (FR 2010)*, Kyoto, Japan, December 2009.

[92] T. H. Fanning and J. W. Thomas, *Advances in Coupled Safety Modeling Using Systems Analysis and High-Fidelity Methods*, ANL-GENIV-134, Argonne National Laboratory, May 2010.

[93] J. W. Thomas, T. H. Fanning, R. Vilim and L. L. Briggs, "Validation of the Integration of CFD and SAS4A/SASSYS-1: Analysis of EBR-II Shutdown Heat Removal Test 17," in *Proc. 2012 Int'l Congress on Advances in Nuclear Power Plants (ICAPP '12)*, Chicago, IL, June 2012.

[94] "reStructuredText: Markup Syntax and Parser Component of Docutils," 2016. [Online]. Available: http://docutils.sourceforge.net/rst.html. [Accessed 28 June 2018].

[95] "LaTeX Documentation," The LaTeX Project, [Online]. Available: https://www.latex-project.org/help/documentation/. [Accessed 28 June 2018].

[96] "trac: Integrated SCM & Project Management," The trac Project, [Online]. Available: https://trac.edgewall.org/wiki/TracGuide. [Accessed 28 June 2018].

[97] *ISO/IEC 1539-1:2004: Programming Languages — Fortran — Part 1: Base Language*, International Standards Organization, 2004.

[98] *ISO/IEC 9899:1999: Programming Languages — C*, International Standards Organization, 1999.

[99] *ISO/IEC 14882:2003: Programming Languages — C++*, International Standards Organization, 2003.

[100] "The GNU Fortran Compiler," Free Software Foundation, Inc., [Online]. Available: https://gcc.gnu.org/onlinedocs/gfortran/. [Accessed 28 June 2018].

[101] "NAG Fortran Compiler," Numerical Algorithms Group, [Online]. Available: https://www.nag.com/nag-compiler. [Accessed 28 June 2018].

[102] A. Karahan, "Development of Advanced In-Pin Metallic Fuel Performance Models for SAS4A," in *Trans. American Nuclear Society*, June 2014.

[103] A. Karahan and A. Tentner, "Validation of Advanced Metallic Fuel Models of SAS4A using TREAT M-Series Overpower Test Simulations," in *Int'l Conf. on Fast Reactors and Related Fuel Cycles (FY-17)*, Yekaterinburg, Russia, June 2017.

[104] A. Karahan, "Mechanistic Modeling of Metallic Fuel/Cladding Metallurgical Interactions," in *Trans. American Nuclear Society*, June 2014.

[105] A. M. Tentner, S. H. Kang and A. Karahan, "Advances in the Development of the SAS4A Code Metallic Fuel Models for the Analysis of PGSFR Postulated Severe Accidents," in *Int'l Conf. Fast Reactors and Related Fuel Cycles (FR-17)*, Yekaterinburg, Russian Federation, June 2017.

[106] A. M. Tentner and A. Karahan, "LEVITATE-M: The Fuel Relocation Model of the SAS4A Code for the Analysis of Postulated Severe Accidents in Metal Fuel Sodium Fast Reactors," in *Trans. American Nuclear Society*, Washington, DC, November 2017.

[107] A. J. Brunett and T. H. Fanning, "Uncertainty Quantification in Advanced Reactors: The Coupling of SAS4A/SASSYS-1 with RAVEN and Dakota," in *Proc. of ICAPP 2016*, San Francisco, CA, April 2016.

[108] G. Zhang, A. J. Brunett, T. Sumner, N. Stauff and T. H. Fanning, "Dakota-SAS4A/SASSYS-1 Coupling for Uncertainty Quantification and Optimization Analysis," in *Proc. Int'l Conf. on Mathematics & Computational Methods Applied to Nuclear Science & Engineering*, Jeju, Korea, April 2017.

[109] G. Zhang, T. Sumner and T. H. Fanning, "Uncertainty Quantification of EBR-II Loss of Heat Sink Simulations with SAS4A/SASSYS-1 and DAKOTA," in *Proc. of Int'l Conf. on Fast Reactors and Related Fuel Cycles*, Yekaterinburg, Russia, June 2017.

[110] G. Zhang, K. Zeng, N. Stauff, J. Hou, T. K. Kim and T. H. Fanning, "Uncertainty Quantification of ABR Transient Safety Analysis," in *Proc. 2018 Best Estimate Plus Uncertainty International Conf. (BEPU 2018)*, Lucca, Italy, May 2018.

[111] D. W. Henneke and J. Robinson, *Development/Modernization of an Advanced Non-Light Water Reactor Probabilistic Risk Assessment*, DOEGEHH08325, GE-Hitachi Nuclear Energy Americas LLC, Wilmington, NC, 2017.

[112] A. J. Brunett, D. Grabaskas, M. Bucknor and S. Passerini, "A Methodology for the Integration of Passive System Reliability with Success Criteria in a Probabilistic Framework for Advanced Reactors," in *Proc. Int'l Conf. on Nuclear Engineering (ICONE 24)*, Charlotte, NC, June 2016.

[113] D. Grabaskas, A. J. Brunett and M. Bucknor, "A Methodology for the Integration of a Mechanistic Source Term Analysis in a Probabilistic Framework for Advanced Reactors," in *Proc. Int'l Conf. on Nuclear Engineering (ICONE 24)*, Charlotte, NC, June 2016.

[114] D. Grabaskas, A. J. Brunett and M. Bucknor, "A Methodology for the Development of a Reliability Database for an Advanced Reactor Probabilistic Risk Assessment," in *Proc. Int'l Conf. on Nuclear Engineering (ICONE 24)*, Charlotte, NC, June 2016.

[115] M. Warner, J. Li and J. Hagaman, "Development of Non-LWR PRA Methodologies for an Advanced Non-LWR Technology Using a Risk-Informed Framework," in *Proc. Int'l Conf. on Nuclear Engineering (ICONE 24)*, Charlotte, NC, June 2016.

[116] J. Li and J. Hagaman, "Demonstration of a Non-LWR Success Criteria Methodology in a Probabilistic Framework for Advanced Reactors," in *Proc. Int'l Conf. on Nuclear Engineering (ICONE 24)*, Charlotte, NC, June 2016.

[117] J. E. Hagaman, J. Young, M. Warner, D. W. Henneke and J. Li, "Identification of Research and Development Needs for Non-LWR PRA," in *Proc. Int'l Topical Mtg. on Probabilistic Safety Assessment and Analysis (PSA 2017)*, Pittsburgh, PA, 2017.

[118] Z. Jankovsky, *personal communications*, Sandia National Laboratories, July 2015.

Nuclear Science and Engineering Division
Argonne National Laboratory
9700 South Cass Avenue, Bldg. 208
Argonne, IL 60439-4842

www.anl.gov